Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Blaž Sovdat

Algoritmi za inkrementalno učenje odločitvenih dreves na spremenljivih podatkovnih tokovih

Delo je pripravljeno v skladu s Pravilnikom o podeljevanju Prešernovih nagrad študentom, pod mentorstvom

izr. prof. Zorana Bosnića.

Ljubljana 2013

Zahvala

Najprej hvala mentorju doc. dr. Zoranu Bosniću za usmerjanje, koristne nasvete in komentarje glede dela.

Posebej se zahvaljujem neuradnemu mentorju dr. Blažu Fortuni za konstantno pomoč, nasvete, komentarje glede teksta in kode, dobro voljo in prijateljski odnos.

Izr. prof. dr. Marko Robnik-Šikonja, asist. dr. Martin Vuk, izr. prof. dr. Zoran Bosnić in izr. prof. Uroš Lotrič na FRI ter asist. dr. Andrej Muhič in Jan Rupnik na IJS so mi dali uporabne predloge in komentarje glede inkrementalnih formul in algoritmov za entropijo in Gini indeks.

Hvala vsem ostalim na IJS za izredno prijetno okolje v zadnjem letu, posebej dr. Marku Grobelniku, dr. Blažu in dr. Carolini Fortuna, ki so mi predstavili področje učenja na podatkovnih tokovih.

Sošolcem in prijateljem se zahvaljujem za prijetna zadnja tri leta, posebej Gašperju, Matiji, Matevžu in Borji.

Končno hvala staršema, Blanki in Milanu, in bratu Luku za vso moralno in finančno podporo med študijem.

"Finally I am becoming stupider no more."

—Pál Erdős (1913–1996)

Povzetek

V tem delu podrobno predstavimo glavne ideje za trenutno najbolj popularnimi algoritmi za sprotno učenje odločitvenih dreves in drugih modelov na spremenljivih podatkovnih tokovih.

Začnemo z dokazom Hoeffdingove neenakosti in na podlagi le-te pridemo do splošne metode za vertikalno skaliranje algoritmov strojnega učenja za učenje na podatkovnih tokovih. Uporabo te metode ilustriramo na primeru inkrementalnega učenja klasifikacijskih dreves na spremenljivih podatkovnih tokovih. Te učne algoritme podrobneje opišemo in dokažemo nekaj pripadajočih teoretičnih zagotovil.

Algoritme za inkrementalno učenje odločitvenih dreves implementiramo in damo veliko sliko in primer uporabe naše implementacije.

Predstavimo metode za ocenjevanje uspešnosti učenja in primerjavo algoritmov na spremenljivih podatkovnih tokovih. Izvedemo eksperimente na realnem scenariju napovedovanja porabe električne energije za zvezno državo New York, kjer ilustriramo uporabo implementacije in hkrati uporabo metod za ocenjevanje uspešnosti učenja in primerjavo algoritmov na spremenljivih podatkovnih tokovih.

Originalen prispevek so preproste inkrementalne formule in algoritmi za računanje entropije in Gini indeksa na spremenljivih podatkovnih tokovih.

Ključne besede: strojno učenje, sprotno učenje, odločitvena drevesa, spremenljivi podatkovni tokovi, Hoeffdingova neenakost, evalvacija algoritmov na podatkovnih tokovih

Abstract

This work is detailed presentation of the main ideas behind state-of-the-art algorithms for online learning of decision trees and other models from time-changing data streams.

We begin by proving Hoeffding inequality, which we then use to derive a general method for scaling up machine learning algorithms. We apply this method to scale up classical decision tree learning algorithm, and prove theoretical guarantees for one of the learners.

We implement scaled up decision tree learners and, after giving a rough description of the implementation, illustrate usage on a simple, bootstrapped dataset.

We then turn to methods for assessing stream learning algorithm performance and comparing two algorithms on a single data stream. We apply these methods when performing experiments on a real-world electricity-demand scenario for New York state electricity data, demonstrating usage of both our implementation and evaluation methods.

Original contribution are simple formulas and algorithms for computing entropy and Gini index on time-changing data streams.

Keywords: online learning, machine learning, decision trees, time-changing data streams, Hoeffding inequality, stream learning algorithm evaluation

Kazalo

1 Uvod

"Big Data does not need big machines. It needs big intelligence."

—Paolo Boldi

Strojno učenje je veja računske znanosti, kjer se računalnik uči programov iz podatkov. Tukaj se ukvarjamo z *nadzorovanim učenjem*, kjer na podlagi učne množice označenih primerov $S := \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, pri čemer je oznake y_i generirala neznana funkcija f, iščemo funkcijo h, ki čim bolje aproksimira resnično funkcijo f. Pri tem je vsak vektor $\mathbf{x} \in \mathcal{A}$ predstavljen z vrednostmi n atributov, pri čemer $\mathcal{A} := A_1 \times A_2 \times \ldots \times A_n$ za neprazne množice $A_i \neq \emptyset$ imenujemo *atributni prostor*; pripadajoča oznaka $y \in C$ leži v množici oznak C. Preslikavi h rečemo *hipoteza*. Učenje je iskanje hipoteze $h \in \mathcal{H}$ v prostoru hipotez $\mathcal{H}$, ki se bo "dobro obnesla" na novih primerih. Natančnost hipoteze izračunamo na *testni množici*, ki je različna od učne množice. Rečemo, da hipoteza dobro *generalizira*, če pravilno napove oznake novih primerov. V grobem ločimo naslednja dva tipa učnih problemov:

- kadar je množica C končna, imamo *klasifikacijski* problem,
- kadar je $C \subseteq \mathbb{R}$, so oznake numerične in rečemo, da gre za *regresijski* problem[1].

Preslikavo f, ki je ne poznamo, aproksimiramo s preslikavo h iz hipoteznega prostora

$$\mathcal{H} := \{h \mid h : \mathcal{A} \to C\}$$

vseh modelov[2]. Rečemo, da je učni problem *uresničljiv*, če hipotezni prostor vsebuje pravo funkcijo f; včasih ne vemo ali je naš problem uresničljiv, ker ne poznamo iskane funkcije. (Kadar je $\mathcal{A} = \mathbb{R}^d$ za $d \in \mathbb{N}$ in $C = \mathbb{R}$, in se omejimo na linearne preslikave $h : \mathbb{R}^{d+1} \to \mathbb{R}$ za $h(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + c$, je hipotezni prostor, v tem primeru koeficientov, enak $\mathcal{H} = \mathbb{R}^{d+1}$. Ta hipotezni prostor po definiciji ne vsebuje polinomov višjih stopenj, zato v tem primeru problem učenja poljubnega polinoma višje stopnje ni uresničljiv.)

Iščemo torej hipotezo, ki se najbolje prilega podatkom S:

$$\begin{aligned} h^\star &= \arg\min_{h \in \mathcal{H}} \ \mathbb{P}(h|S) \\ &= \arg\min_{h \in \mathcal{H}} \ \mathbb{P}(S|h)\,\mathbb{P}(h), \end{aligned}$$

pri čemer zadnja enakost sledi zaradi Bayesovega izreka.

Omenimo, da bi lahko definirali $\mathcal{H}$ kot razred vseh Turingovih strojev — vsako izračunljivo funkcijo lahko predstavimo s Turingovim strojem. V splošnem tak problem ni izračunljiv; ne upošteva zahtevnosti učenja; in ne upošteva zahtevnosti računanja naučene hipoteze, ki je v tem primeru Turingov stroj. Ta ideja torej ni praktična.

[1] Izpostavimo, da regresijski učni problem formalno sprašuje po pogojni pričakovani vrednosti $\mathbb{E}[Y|X = \mathbf{x}]$, saj je verjetnost, da smo našli natanko dan $y \in C$, enaka nič.

[2] Formalno je $\mathcal{H}$ prostor vseh možnih modelov. Pri učenju odločitvenih dreves je $\mathcal{H}$ prostor vseh odločitvenih dreves, ki jih lahko zgradimo iz danih atributov.

Namesto napake ponavadi minimiziramo funkcijo izgube $L : C \times C \to \mathbb{R}_0^+$, dano s predpisom

$$L(y, \hat{y}) := \begin{cases} 0, & y = \hat{y}, \\ 1, & y \neq \hat{y}. \end{cases}$$

Naj bo $\mathbb{P}(X, Y)$ apriorna verjetnostna porazdelitev primerov. Potem iščemo hipotezo $h \in \mathcal{H}$, ki minimizira pričakovano izgubo čez vse učne primere S. Tako definiramo *posplošeno izgubo* kot

$$L_G(h) := \sum_{(\mathbf{x},y)\in S} L(y, h(\mathbf{x}))\, \mathbb{P}(\mathbf{x}, y),$$

in iščemo hipotezo $h^\star$, ki minimizira pričakovano posplošeno izgubo:

$$h^\star = \arg\min_{h\in\mathcal{H}} L_G(h).$$

Ker ne poznamo $\mathbb{P}(\mathbf{x}, y)$, lahko učenec generalizacijsko izgubo le oceni z *empirično izgubo* na množici primerov E kot

$$L_E(h) := \frac{1}{|E|} \sum_{(\mathbf{x},y)\in E} L(y, h(\mathbf{x})),$$

kar pomeni, da je ocenjeno najboljša hipoteza $\hat{h}^\star$ tista, ki minimizira empirično izgubo:

$$\hat{h}^\star = \arg\min_{h\in\mathcal{H}} L_E(h).$$

Hipotezni prostor $\mathcal{H}$ je ponavadi ogromen, kar pokažemo z naslednjim primerom. Naj bo $n \in \mathbb{N}$ naravno število, naj bo $\mathcal{A} := \{0, 1\}^n$ naš atributni prostor in naj bo $C := \{0, 1\}$ množica oznak. Nadalje recimo, da se želimo učiti Boolove funkcije. Potem je hipotezni prostor množica vseh preslikav $f : \mathcal{A} \to C$, zato — kot se lahko prepričamo s preprostim kombinatornim sklepom — velja

$$\begin{aligned} |\mathcal{H}| &= |C|^{|\mathcal{A}|} \\ &= 2^{2^n}. \end{aligned}$$

Za $n = 7$ binarnih atributov to pomeni $340\,282\,366\,920\,938\,463\,463\,374\,607\,431\,768\,211\,456 \approx 10^{38}$ možnih modelov, za $n = 10$ pa je $|\mathcal{H}| \approx 10^{308}$, kar je veliko več kot ocenjeno število atomov v vidnem vesolju.[3] Iskanje optimalne hipoteze — kar v splošnem pomeni preiskovanje celotnega hipoteznega prostora — je torej brezupno in ponavadi učni algoritmi požrešno preiskujejo hipotezni prostor.

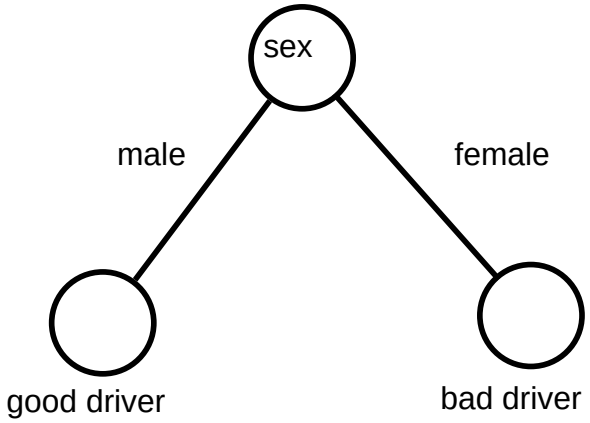
V tem delu se bomo ukvarjali z *inkrementalnim učenjem odločitvenih dreves* na *spremenljivih podatkovnih tokovih*. Odločitveno drevo klasificira primer tako, da ga sortira iz *korena* (vozlišče brez staršev) v ustrezen *list* (vozlišče brez otrok). Vsako *vozlišče* (vozlišče z otroki) drevesa poda primer otroku glede na vrednost določenega atributa za ta učni primer. Posamezne veje iz vozlišča ustrezajo vrednostim atributa na katerem vozlišče testira. Primer klasificiramo tako, da (i) začnemo v korenu odločitvenega drevesa, (ii) testiramo ustrezen atribut in se pomaknemo po veji, ki ustreza vrednosti tega atributa; (iii) če trenutno vozlišče ni list, ponovimo (i) na poddrevesu, ki ga vpenja trenutno vozlišče, sicer pa napovemo oznako, ki se v tem listu največkrat pojavi. (Odločitveno drevo lahko predstavimo kot množico odločitvenih pravil: vsako pot od korena do lista predstavimo kot konjunkcijo testov in napovemo oznako, ki se v tem listu največkrat pojavi.)

Slika 1.1 prikazuje primer (patriarhalnega) klasifikacijskega drevesa, ki glede na spol loči voznike v dobre in slabe. Če bi se to drevo res naučili iz podatkov, bi ga interpretirali kot: moški so dobri vozniki, ženske pa ne. *Interpretabilnost* in *razumljivost* sta veliki prednosti odločitvenih dreves pred ostalimi vrstami modelov. Drevo vedno uporabi najpomembnejši atribut v korenu — to se lepo vidi iz drevesa, ki napoveduje kdo je preživel na Titanicu.

V splošnem ločimo dva tipa odločitvenih dreves:

- *klasifikacijska* drevesa, ki napovedujejo diskretno oznako,

[3] Ocenjeno število atomov v vidnem vesolju leži med 10^{78} in 10^{82}.



Slika 1.1: Primer preprostega klasifikacijskega drevesa.

- *regresijska* drevesa, ki napovedujejo realno število.

Odločitvena drevesa iz prejšnjega odstavka so torej klasifikacijska. Izpostavimo, da uporabljajo le *diskretne atribute* — ti imajo končno zalogo vrednosti (npr. oseba je lahko moškega ali ženskega spola; torej je spol primer diskretnega atributa). Druga vrsta so *numerični atributi* — ti zavzamejo poljubno realno število in jih je med učenjem potrebno *diskretizirati*. Odločitvena drevesa na numeričnem atributu A najpogosteje testirajo $A < x$ in $A \geq x$ za nek $x \in \mathbb{R}$; torej je opis iz prejšnjega odstavka še vedno veljaven, če se delamo, da je A *binaren atribut* in preslikamo realna števila manjša od x v prvo vrednost in vsa ostala realna števila v drugo vrednost.

Zaenkrat nismo povedali kako se odločitveno drevo naučimo, ampak le, kako ga *uporabljamo* za napovedovanje. S problemom učenja se ukvarjamo v preostalem delu naloge.

Več podrobnosti najdemo v [Russell and Norvig, 2012], učbeniku [Kononenko and Robnik-Šikonja, 2010] in odličnem članku [Domingos, 2012]. Dobra referenca je tudi [Mitchell, 1997].

1.1 Motivacija

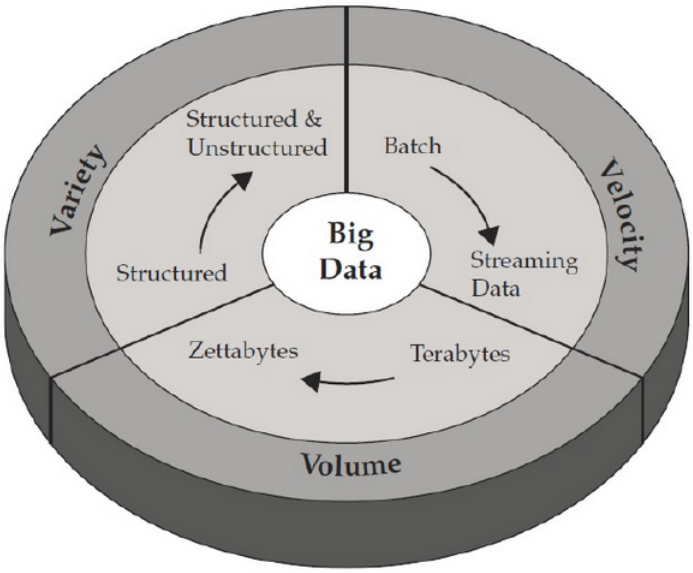
Rekli smo, da se v strojnem učenju računalnik uči programov iz podatkov. Glede na to, da programe pišemo ljudje, ni jasno, zakaj potrebujemo strojno učenje. Zanimivo je, da verjetno nihče ne zna ročno napisati algoritma, ki bi na sliki prepoznal človeški obraz — ta problem danes rešujemo skoraj izključno z algoritmi strojnega učenja[4]. Nekateri ostali primeri uspešne uporabe strojnega učenja so napovedovanje ocene filma, ocenjevanje pomembnosti spletne strani glede na iskalni niz in uporabnikovo zgodovino, napovedovanje prijateljstev na socialnih omrežjih, prepoznavanje izrazov na obrazu in zaznavanje vdorov v epoštni račun.

V prejšnjih dveh desetletjih so se v strojnem učenju ukvarjali predvsem s paketnim učenjem na relativno majhnih podatkih, kjer so vsi podatki na voljo vnaprej in jih lahko spravimo v delovni pomnilnik. Paketni učenci med gradnjo napovednega modela tipično večkrat procesirajo podatke. Razlog za to je predpostavka, da so primeri generirani naključno po neki stacionarni porazdelitvi. Lep primer je algoritem TDIT [Quinlan, 1993] za učenje odločitvenih dreves. V nekaterih domenah, kot je medicinska diagnostika, je podatkov velikokrat premalo — tako ni redko, da se napovedne modele za raka na dojki gradi na 80 učnih primerih.

Vstopamo v obdobje "Big Data". Izraz pomeni eksplozijo dosegljivih podatkov, ki jih ne moremo obdelati z obstoječo infrastrukturo in algoritmi (glej sliko 1.2). Od leta 2012 naprej naj bi vsak dan generirali okoli $2.5 \cdot 10^{18}$ bajtov podatkov [Bifet, 2013],[5] ki jih zaradi omenjenih omejitev slabo

[4]Primer je hiter in učinkovit Viola-Jones algoritem [Viola and Jones, 2001, Jones and Viola, 2001], ki se med drugim uporablja v digitalnih kamerah.

[5]To med drugim pomeni, da *vsako sekundo* v povprečju generiramo okoli 29TB podatkov!



Slika 1.2: Karakterizacija Big Data: Volume, Velocity, Variety [Grobelnik, 2013].

izkoriščamo. V nekaterih domenah vir, ki generira podatke, ni stacionaren, primeri pa prihajajo hitro in zvezno in za praktične namene neomejeno — primer so iskalni zahtevki na Google (povprečno 5 134 000 000 zahtevkov na dan v 2012),[6] zahtevki na Twitterju (aprila 2010 okoli 106 000 000 uporabnikov, 3 000 000 000 API zahtevkov dnevno, in 600 000 000 iskalnih zahtevkov na dan [Bifet, 2013]) in Facebooku (3M sporočil na 20 minut),[7] članki, komentarji in kliki na novičarskih portalih kot sta Bloomberg (200 klikov na sekundo) in NY Times (50GB logov, 6M unikatnih uporabnikov, 50M klikov na dan),[8] in meritve v senzorskih omrežjih, kjer se podatki izgubijo, če jih ne obdelamo takoj. Take podatke najlažje modeliramo kot podatkovne tokove.

V domenah iz prejšnjega odstavka klasični pristopi strojnega učenja odpovejo, kar nas motivira k iskanju novih pristopov. V tem delu se bomo ukvarjali z algoritmi, ki procesirajo vsak primer posebej in sproti gradijo ter po potrebi spreminjajo odločitveno drevo. Pri tem lahko trenutni model kadarkoli uporabimo.

1.2 Organizacija dokumenta

Delo je razdeljeno na tri dele — teoretične osnove, implementacija, in dodatki — ki so opisani v sledečih podrazdelkih. Naslednje poglavje je opis osnovnih konceptov in pregled dela.

1.2.1 Teoretične osnove

Začnemo z dokazom Hoeffdingove neenakosti in njeno uporabo za učenje na podatkovnih tokovih (poglavje 3). Nadaljujemo z opisom splošne metode za učenje na podatkovnih tokovih (poglavje 4) in dvema različicama algoritmov za učenje odločitvenih dreves. Zatem predstavimo inkrementalne formule in algoritme za računanje entropije in Gini indeksa na spremenljivih podatkovnih tokovih (poglavje 5).

[6] Vir: `http://www.statisticbrain.com/google-searches/`.
[7] Vir: `http://www.statisticbrain.com/facebook-statistics/`.
[8] Na projektu XLike (`http://xlike.org/`) analizirajo tok z nekaj 10 članki na sekundo [Grobelnik, 2013].

1.2.2 Implementacija

V tem delu naprej podamo veliko sliko implementacije (poglavje 6) in podrobneje opišemo diskretizacijo numeričnih atributov. Zatem navedemo primer uporabe (poglavje 7) na preprostih podatkih in demonstriramo glavne funkcionalnosti: (i) pisanje konfiguracije, (ii) priprava podatkov v pravem formatu, in (iii) uporaba implementacije v C++ in Javascript. Nadaljujemo s hitrim pregledom metod za ocenjevanje uspešnosti učenja in primerjanje algoritmov na podatkovnih tokovih (poglavje 8). Le-te uporabimo za izvedbo eksperimentov (poglavje 9) na podatkih o porabi energije v zvezni državi New York, ki jih od leta 2001 dnevno objavlja neodvisni sistemski operater distribucijskega omrežja električne energije.

1.2.3 Dodatki

Najprej navedemo pregled obstoječih implementacij za učenje na podatkovnih tokovih (dodatek A). Zaključimo s seznamom uporabljenih podatkov in podrobnejšim opisom podatkov za realen scenarij (dodatek B).

1.3 Originalni prispevki

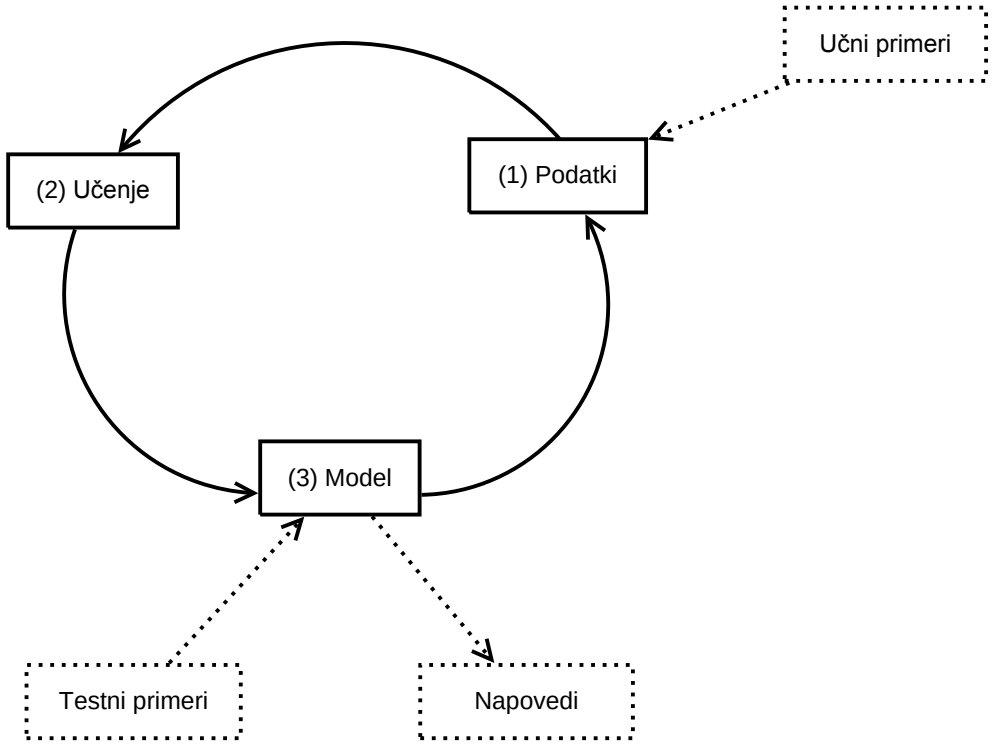
Poglavje 5 je v celoti plod lastnih razmišljanj. Dokažemo preproste inkrementalne formule za informacijsko-teoretično entropijo, ki so v našem kontekstu uporabne za inkrementalno računanje informacijskega dobitka pri inkrementalnem učenju odločitvenih dreves in pravil. Na podlagi teh formul navedemo inkrementalne algoritme za računanje entropije na spremenljivih podatkovnih tokovih. Prilagajanje potencialnim spremembam dosežemo z uporabo faktorjev pozabljanja in drsečih oken. Podobno dokažemo preproste inkrementalne formule za računanje Gini indeksa, ki so uporabne za ocenjevanje atributov pri inkrementalnem učenju klasifikacijskih dreves in pravil. Na podlagi teh formul navedemo inkrementalne algoritme za računanje Gini indeksa na spremenljivih podatkovnih tokovih. Tudi ti algoritmi uporabljajo drseča okna in faktorje pozabljanja.

2 Osnovne definicije in pregled dela

V tem poglavju navedemo glavne razlike med paketnim učenjem in učenjem na podatkovnih tokovih. Drugi razdelek je pregled relevantnega dela.

2.1 Osnovne definicije in koncepti

Učenje na podatkovnih tokovih je poseben primer sprotnega učenja (angl. online learning). Glavna razlika v primerjavi s paketnim učenjem je v tem, da pri učenju na podatkovnih tokovih v model vključimo nove informacije z majhnimi spremembami modela, brez da bi se model na novo naučili. Medtem ko je paketno učenje končen proces, ki se začne z zbiranjem podatkov in konča z modelom, ki se v bližnji prihodnosti ne bo spreminjal, je sprotno učenje nekončen proces, ki se začne s prihodom učnih primerov in traja dokler imamo na voljo podatke za učenje. Sprotno učenje je torej proces, ki združuje faze zbiranja podatkov, učenja, in napovedovanja v zvezen cikel (slika 2.1).



Slika 2.1: Zvezen cikel učenja na podatkovnih tokovih [Bifet et al., 2009].

Za podatkovne tokove so zahteve drugačne kot pri klasičnem okolju [Bifet et al., 2009]:

1. procesiraj po en učni primer naenkrat in vsak učni primer poglej največ enkrat,

2. uporabljaj omejeno količino delovnega pomnilnika,

3. delaj v omejenem času,

4. bodi pripravljen, da lahko začneš kadarkoli napovedovati.

Na sliki 2.1 algoritem dobi naslednji učni primer (zahteva 1), ga procesira, in posodobi interne podatkovne strukture s čim manjšo porabo delovnega pomnilnika (zahteva 2) in čim hitreje (zahteva 3); zatem je pripravljen na naslednji primer in na napovedovanje (zahteva 4).

Glavne razlike med klasičnimi podatkovnimi zbirkami in podatkovnimi tokovi lahko povzamemo v naslednjih točkah:

- zapisi na podatkovnih tokovih prihajajo sproti,
- pri podatkovnih tokovih nimamo nadzora nad vrstnim redom in hitrostjo prihajanja zapisov v podatkovnem toku,
- podatkovni tokovi so za vse praktične namene neomejeni,
- kadar procesiramo zapis podatkovnega toka, ga ali zavržemo ali arhiviramo — delovni spomin je zanemarljivo majhen v primerjavi z velikostjo podatkovnega toka, zato do arhiviranega zapisa nimamo poceni dostopa.

Tabeli 2.1 in 2.2 prikazujeta glavne razlike med paketnim in sprotnim učenjem in med podatkovnimi bazami in podatkovnimi tokovi.

	Paketno učenje	Učenje na podatkovnih tokovih
Velikost podatkov	Končna množica podatkov	Zvezen tok podatkov
Porazdelitev podatkov	Neodvisna, enakomerna	Odvisna, neenakomerna
Spremenljivost podatkov	Nespremenljivi	Spremenljivi
Gradnja modela	Paketna	Inkrementalna
Stabilnost modela	Statičen model	Spremenljiv model
Vrstni red primerov	Neodvisen od podatkov	Odvisen (nimamo nadzora)

Tabela 2.1: Razlike med paketnim učenjem in učenjem na podatkovnih tokovih [Gama et al., 2013].

	Podatkovne baze	Podatkovni tokovi
Dostop	Naključni	Zaporedni
Število prehodov	Več prehodov	En prehod
Čas procesiranja	Neomejen	Omejen
Pomnilnik	Neomejen	Fiksen
Rezultat	Natančen	Približen
Porazdeljeni podatki	Ne	Da

Tabela 2.2: Glavne razlike med klasičnimi podatkovnimi bazami in podatkovnimi tokovi [Gama, 2012].

Da algoritem lahko procesira hitre in neomejene podatkovne tokove, potrebuje naslednje lastnosti [Hulten et al., 2001, Ikonomovska, 2012]:

- model naj se nauči z enim sprehodom čez podatke,
- naj ima majhen, potencialno konstanten, čas za procesiranje posameznih primerov,
- naj uporablja fiksno količino glavnega pomnilnika, neodvisno od velikosti podatkovnega toka,
- naj sproti inkorporira novo informacijo v obstoječ model,

- naj se prilagaja spremembam v podatkih.

Za več glej [Ikonomovska, 2012] in [Gama, 2012, Gama et al., 2013]. Naslednji razdelek je (zelo nepopoln) pregled obstoječega dela na temo učenja na podatkovnih tokovih, kjer navedemo množico praktično uspešnih algoritmov, ki izpolnjujejo pogoje iz prejšnjih odstavkov.

2.2 Pregled dela

Prvi rezultati s področja sprotnega učenja segajo v drugo polovico prejšnjega stoletja, ko je [Rosenblatt, 1958] predstavil Perceptron. Tukaj ignoriramo večino teoretičnega dela in navedemo praktično uspešne pristope za učenje na podatkovnih tokovih.

2.2.1 Odločitvena drevesa

Prvi zares uspešen algoritem za inkrementalno učenje odločitvenih dreves sta predstavila [Domingos and Hulten, 2000], kjer je glavna ideja v uporabi Hoeffdingove neenakosti pri iskanju atributa za razcep lista; algoritem, imenovan VFDT, je sicer inkrementalna varianta klasičnega ID3 algoritma. Algoritem VFDT so [Hulten et al., 2001] razširili tako, da se prilagaja spremembam v podatkih — v grobem gre za VFDT algoritem z drsečim oknom fiksne velikosti, kjer spremenimo zadostne statistike za izračun ocen atributov vsem vozliščem, kadar v okno pride nov primer in kadar iz okna pade najstarejši primer. Ta algoritem se imenuje CVFDT.

Sledila je eksplozija člankov, v katerih so avtorji predstavili mnogo različic algoritmov VFDT in CVFDT. Omenimo [Jin and Agrawal, 2003], ki namesto Hoeffdingovega predstavita t.i. normalni test, ki izkorišča mere nečistoče. Originalno VFDT v listih uporablja večinski klasifikator in ne podpira numeričnih atributov; to so rešili [Gama et al., 2003] z varianto VFDTc, ki uporablja Naivni Bayesov klasifikator v listih in potratno diskretizacijo numeričnih atributov.

Bifet in sod. [Bifet et al., 2009, 2010a, 2012] so objavili prve metode za učenje ansamblov in za večznačno klasifikacijo [Read et al., 2011, 2012]. Veliko večino teh pristopov so implementirali v odprtokodnem Javanskem paketu MOA [Bifet et al., 2010b].

Končno sta [Ikonomovska and Gama, 2008] predstavila algoritem FIMT za inkrementalno učenje regresijskih dreves, ki so ga [Ikonomovska et al., 2009, 2011, Ikonomovska, 2012] razširili za prilagajanje spremembam v podatkih (FIRT-DD in FIMT-DD) in za večciljno napovedovanje (FIMT-MT).

Obstajajo tudi vzporedni algoritmi za učenje dreves na podatkovnih tokovih [Ben-Haim and Tom-Tov, 2010].

Glej [Bifet et al., 2011] za podrobnejši opis delovanja nekaterih omenjenih pristopov in za nadaljnje reference.

2.2.2 Razvrščanje

Pri problemu razvrščanja moramo podobne objekte grupirati v enake gruče, različne objekte pa v različne gruče, tako da so razlike med objekti znotraj gruč majhne, razlike med objekti iz različnih gruč pa velike. Problem razvrščanja na podatkovnih tokovih sprašuje po konsistentno dobrem gručenju primerov, ki jih je učenec videli do sedaj; pri tem želimo, da učenec hitro procesira nove primere in porabi čim manj delovnega pomnilnika [Gama, 2012].

Algoritem BIRCH [Zhang et al., 1996] je [Novak, 2009, 2008] razširil in uporabil na tekstovnih podatkih; varianto BIRCHa so predstavili tudi [Aggarwal et al., 2003]. Inkrementalno varianto k-means, imenovano VFKM, sta predstavila [Domingos and Hulten, 2001a]; različico prilagojeno tekstovnim podatkom implementira tudi [Brank, 2013].

2.2.3 Ostalo

Domingos in Hulten sta idejo iz podrazdelka 2.2.1 posplošila [Domingos and Hulten, 2001a,b, 2003] in jo uporabila za inkrementalne variante k-means razvrščanja (VFKM), Bayesovih mrež (VFBN) in EM algoritma (VFEM). Izdala sta tudi odprtokodni Cjevski paket VFML [Hulten and Domingos,

2003]. Svoje delo glede učenja na podatkovnih tokovih sta povzela v neobjavljenem članku [Hulten et al., 2005].

Pred kratkim so [Gama and Kosina, 2011, Kosina and Gama, 2012b,a, Almeida et al., 2013] predstavili algoritme za učenje odločitvenih pravil na spremenljivih podatkovnih tokovih, kjer za razširjanje pravil uporabljajo Hoeffdingovo neenakost — kot Domingos in Hulten za drevesa — in svoj algoritem imenujejo VFDR. (Izpostavimo, da so inkrementalno učenje odločitvenih pravil predstavili že [Schlimmer and Granger, 1986].)

Ocenjevanje uspešnosti učenja in primerjava algoritmov na podatkovnih tokovih zahteva nove pristope. V grobem ločimo oceno napake z izločanjem testnih primerov (angl. holdout error estimation) in oceno pričakovane napake (angl. prequential error estimation), za primerjavo dveh algoritmov na toku pa ponavadi uporabljamo Q-statistiko [Gama et al., 2009, 2013].

Omenimo še nalogi [Kešpret, 2012] in [Demšar, 2012], ki se dotikata podatkovnih tokov.

Del I

Teoretične osnove

3 Verjetnostne neenakosti

V tem poglavju povzamemo dokaz Hoeffdingove neenakosti,[1] ki v času tega pisanja predstavlja osnovo za (nekatere) najbolj uspešne algoritme za učenje na podatkovnih tokovih.

Na podlagi Hoeffdingove neenakosti povzamemo izpeljavo Hoeffdingovega testa in ilustriramo njegovo uporabo na primeru učenja pričakovanja omejene naključne spremenljivke. Za popolnost izpeljemo še t.i. normalni test — edini alternativen test pri učenju odločitvenih dreves, ki pa se v praksi ne uporablja.

Zaključimo z enostavno primerjavo testov — oba sta neodvisna od porazdelitve, toda Hoeffdingov test je bolj splošen in zato bolj konzervativen od normalnega testa. (Pod "konzervativen" je mišljena velikost vzorca, potrebna, da dosežemo dano stopnjo zaupanja.)

Pri dokazu Hoeffdingove neenakosti sledimo originalnemu članku [Hoeffding, 1963] z rahlo spremenjeno notacijo. (Razumljiv dokaz Hoeffdingove neenakosti najdemo tudi v [Green, 2013] in [Knuth, 2011].)

3.1 Hoeffdingova neenakost

V tem razdelku povzamemo izpeljavo Hoeffdingove neenakosti (angl. Hoeffding bound), včasih znane kot aditivna neenakost Chernoffa (angl. additive Chernoff bound). Je neodvisna od porazdelitve in nam v grobem da zgornjo mejo za verjetnost, da vsota n nedovisnih omejenih naključnih spremenljivk odstopa od svojega pričakovanja za določeno vrednost $n\epsilon$ za $\epsilon > 0$. Je poseben primer Azuma-Hoeffdingove neenakosti in poseben primer McDiarmidove neenakosti [Motwani and Raghavan, 1995].

Te neenakosti so manifestacije fenomena koncentracije mere (angl. concentration of measure phenomenon). Recimo, da imamo n omejenih naključnih spremenljivk $X_i = O(1)$, ki so med sabo dovolj šibko korelirane. Naj bo $S := X_1 + X_2 + \ldots + X_n$ vsota teh naključnih spremenljivk. Potem trivialno velja $S = O(n)$. Fenomen koncentracije mere pravi, da se vsota naključnih spremenljivk ostro koncentrira v veliko ožjem intervalu, tipično $S = O(\sqrt{n})$. Ta fenomen je vrednoten z neenakostmi visokih deviacij, ki dajo zgornje meje — tipično eksponentne narave — za verjetnost, da taka skombinirana naključna spremenljivka značilno odstopa od svojega pričakovanja. Fenomen velja tudi za splošne nelinearne funkcije $F(X_1, X_2, \ldots, X_n)$ teh naključnih spremenljivk, če je F dovolj regularna. Intuicija je, da dovolj šibko korelirane naključne spremenljivke težko "delujejo skupaj" in se "tepejo" med sabo, zato ne morejo potegniti vsote, ali bolj splošne kombinacije $F(X_1, X_2, \ldots, X_n)$, predaleč od pričakovanja. Tukaj je neodvisnost ključna: rezultati koncentracije mere ponavadi odpovejo, če so X_i preveč korelirane. Torej se verjetnost, da smo vsaj λ standardnih odklonov od pričakovanja manjša kot $C\exp(-c\lambda^2)$ za $c, C > 0$. Bolj točno, odstopamo $O(\log^{1/2} n)$ standardnih odklonov od pričakovanja z veliko verjetnostjo in $O(\log^{1/2+\epsilon} n)$ standardnih odklonov od pričakovanja s "pošastno veliko" verjetnostjo [Tao, 2011].

Glavna ideja dokaza Hoeffdingove neenakosti je naslednja. Verjetnost $\mathbb{P}(S - \mathbb{E}[S] \geq n\epsilon)$ je enaka pričakovani vrednosti indikatorske funkcije $\mathbb{I}[S - E[S] \geq n\epsilon]$, ta pa je od zgoraj omejena z $\exp(c(S - \mathbb{E}[S] - n\epsilon))$ za vsak $c > 0$. Torej je dovolj, da omejimo $\mathbb{E}[\exp(c(S - \mathbb{E}[S] - n\epsilon))]$.

[1] Neenakost je leta 1963 dokazal Wassily Hoeffding (1914–1991), ameriški statistik finskega rodu, ki je med drugim eden od začetnikov U-statistike [Fisher and Van Zwet, 2008].

Izrek 1 ([Hoeffding, 1963])**.** *Naj bo $S := X_1 + X_2 + \ldots + X_n$ vsota neodvisnih omejenih naključnih spremenljivk $a_i \leq X_i \leq b_i$ in naj bo $\epsilon > 0$ poljubno realno število. Potem velja*

$$\mathbb{P}(S - \mathbb{E}[S] \geq n\epsilon) \leq \exp\left(-2n^2\epsilon^2 \Big/ \sum_{i=1}^{n}(b_i - a_i)^2\right). \tag{3.1}$$

Dokaz. Naj bo $c > 0$ parameter, ki ga bomo izbrali kasneje. Za lažje sledenje argumentu naj bo $c_i := c \cdot (b_i - a_i)$ in naj bo $p_i := (\mu_i - a_i)/(b_i - a_i)$, pri čemer je $\mu_i := \mathbb{E}[X_i]$. Opazimo, da velja

$$\mathbb{P}(S - \mathbb{E}[S] \geq n\epsilon) = \mathbb{E}[\mathbb{I}[S - \mathbb{E}[S] \geq n\epsilon]] \leq \mathbb{E}[\exp\left(c(S - \mathbb{E}[S] - n\epsilon\right)].$$

Zaradi neodvisnosti naključnih spremenljivk X_i velja velja

$$\mathbb{E}[\exp\left(c \cdot (S - \mathbb{E}[S] - n\epsilon)\right)] = \exp\left(-cn\epsilon\right) \prod_{i=1}^{n} \mathbb{E}[\exp\left(c \cdot (X_i - \mu_i)\right)].$$

Po lemi 1 imamo

$$\begin{aligned}\mathbb{E}\left[\exp(c(X_i - \mu_i))\right] &\leq \exp(-c\mu_i)\frac{b_i - \mu_i}{b_i - a_i}\exp(ca_i) + \exp(-c\mu_i)\frac{\mu_i - a_i}{b_i - a_i}\exp(cb_i) \\ &= \exp(L_i(c)),\end{aligned}$$

pri čemer postavimo $L_i(c) := -c_i p_i + \log(1 - p_i + p_i \exp(c_i))$.

Ker hočemo čim boljšo mejo, rešujemo $L_i'(c) = 0$. Prva dva odvoda sta

$$\begin{aligned}L_i'(c) &= -p_i + \frac{p_i \exp(c_i)}{1 - p_i + p_i \exp(c_i)} \\ &= -p_i + \frac{p_i}{(1 - p_i)\exp(-c_i) + p_i}\end{aligned}$$

in

$$L_i''(c) = \frac{p_i(1 - p_i)\exp(-c_i)}{\left((1 - p_i)\exp(-c_i) + p_i\right)^2}.$$

Rešitev $L_i'(c) = 0$ je očitno $c_i = 0$. Drugi odvod bo imel največjo vrednost pri $c_i = 0$; dobljeni izraz $p_i(1 - p_i)$ pa bo največji pri $p_i = 1/2$. Zato velja ocena $L_i''(c) \leq 1/4$.

Taylorjeva vrsta funkcije L v okolici točke x_0 je

$$L(x_0) = \sum_{n \geq 0} \frac{L^{(n)}(x_0)}{n!}\left(c_i - x_0\right)^n,$$

od koder ob uporabi $L_i''(c) \leq 1/4$ sklepamo na

$$L_i(c) \leq L_i(0) + L_i'(0)c_i + c_i^2/8.$$

Iz prejšnjih neenakosti sledi

$$\mathbb{P}(S - \mathbb{E}[S] \geq n\epsilon) \leq \exp\left(\frac{1}{8}c^2 \sum_{i=1}^{n}\left(b_i - a_i\right)^2 - cn\epsilon\right).$$

Funkcija v eksponentu ima minimum, kadar velja

$$\frac{\partial}{\partial c}\left(\frac{1}{8}c^2 \sum_{i=1}^{n}\left(b_i - a_i\right)^2 - cn\epsilon\right) = 0,$$

kar se zgodi pri

$$c = 4n\epsilon \Big/ \sum_{i=1}^{n}\left(b_i - a_i\right)^2,$$

od koder z nekaj računanja sledi

$$\mathbb{P}(S - \mathbb{E}[S] \geq n\epsilon) \leq \exp\Big(-2n^2\epsilon^2 \Big/ \sum_{i=1}^{n} \big(b_i - a_i\big)^2 \Big),$$

kar vzpostavi neenakost. □

Lema 1 ([Hoeffding, 1963])**.** *Naj bo X naključna spremenljivka z zalogo $a \leq X \leq b$ in naj bo $c \in \mathbb{R}$ poljubno realno število. Tedaj velja*

$$\mathbb{E}\left[\exp(cX)\right] \leq \frac{b - \mathbb{E}[X]}{b - a} \exp(ca) + \frac{\mathbb{E}[X] - a}{b - a} \exp(cb).$$

Dokaz. Naj bo $a \leq X \leq b$ naključna spremenljivka. Ker je funkcija $\exp(X)$ konveksna, jo navzgor omejuje premica od $X = a$ do $X = b$ in za vsak $c \in \mathbb{R}$ velja

$$\exp(cX) \leq \frac{b - X}{b - a} \exp(ca) + \frac{X - a}{b - a} \exp(cb).$$

Lema sledi, če vzamemo pričakovanje obeh strani. □

Posledica 1. *Naj bo $S := X_1 + X_2 + \ldots + X_n$ vsota neodvisnih omejenih naključnih spremenljivk $a \leq X_i \leq b$ in naj bo $\epsilon > 0$ poljubno realno število. Potem velja*

$$\mathbb{P}(S - \mathbb{E}[S] \geq n\epsilon) \leq \exp\left(-2n\epsilon^2/R^2\right), \tag{3.2}$$

pri čemer je $R = b - a$.

Posledica 2. *Naj bo $S := X_1 + X_2 + \ldots + X_n$ vsota neodvisnih omejenih naključnih spremenljivk $a_i \leq X_i \leq b_i$ in naj bo $\epsilon > 0$ poljubno realno število. Potem velja*

$$\mathbb{P}(|S - \mathbb{E}[S]| \geq n\epsilon) \leq 2\exp\left(-2n^2\epsilon^2 \Big/ \sum_{i=1}^{n} (b_i - a_i)^2 \right). \tag{3.3}$$

Posledica 3. *Naj bo $S := X_1 + X_2 + \ldots + X_n$ vsota neodvisnih omejenih naključnih spremenljivk $a \leq X_i \leq b$ in naj bo $\epsilon > 0$ poljubno realno število. Potem velja*

$$\mathbb{P}(|S - \mathbb{E}[S]| \geq n\epsilon) \leq 2\exp\left(-2n\epsilon^2/R^2\right), \tag{3.4}$$

pri čemer je $R = b - a$.

(Geer navaja Hoeffdingovo neenakost za odvisne naključne spremenljivke [Geer, 2002].)

3.2 Hoeffdingov test

Domingos in Hulten [Domingos and Hulten, 2000] izpeljeta test — recimo mu *Hoeffdingov test* — na podlagi Hoeffdingove neenakosti.

Recimo, da smo naredili n meritev $X_1, X_2, \ldots, X_n$ omejene naključne spremenljivke $a \leq X \leq b$ in ocenili vzorčno povprečje $\overline{\mu} := (X_1 + X_2 + \ldots + X_n)/n$. Naj bo fiksno realno število $\delta \in (0, 1)$ želena zgornja meja za verjetnost za napako. Po Hoeffdingovi neenakosti je $\exp(2n\epsilon^2/R^2)$ zgornja meja za verjetnost, da je $\overline{\mu}$ manjše od $\mathbb{E}[X] - n\epsilon$. Torej verjetnost za napako ne sme biti večja od δ. Zato rešujemo

$$\begin{aligned} \exp\left(-2n\epsilon^2/R^2\right) \leq \delta &\iff -2n\epsilon^2/R^2 \leq \log\delta \\ &\iff \epsilon \geq \sqrt{\frac{R^2\log(1/\delta)}{2n}}. \end{aligned} \tag{3.5}$$

Potem je z verjetnostjo vsaj $1-\delta$ resnično pričakovanje vsaj $\overline{\mu}-\epsilon$.

Pri učenju na podatkovnem toku želimo z določeno verjetnostjo — stopnjo zaupanja — izbrati najboljši atribut pri dani hevristiki. Na število učnih primerov lahko gledamo kot na velikost vzorca, na $\overline{G}(A)$ pa kot na vzorčno oceno. Ker želimo "čim prej" določiti najboljši atribut, v enačbi (3.5) vzamemo najmanjši tak parameter:

$$\epsilon = \sqrt{\frac{R^2 \log(1/\delta)}{2n}}. \tag{3.6}$$

Naj bosta A_1 in A_2 atributa z največjima hevrističnima ocenama $\overline{G}(A_1)$ in $\overline{G}(A_2)$ iz vzorca, in naj bo ϵ izračunan po enačbi (3.6). Potem je A_1 resnično najboljši atribut z verjetnostjo vsaj $1-\delta$, če velja $\overline{G}(A_1) - \overline{G}(A_2) > \epsilon$. (Takrat je razlika večja od nič, ker je $G(A_1) \geq \overline{G}(A_1) - \epsilon$.)

3.3 Normalni test

Jin in Agrawal [Jin and Agrawal, 2003] predlagata manj konzervativen test, ki izkorišča lastnosti mer nečistoče. Naj bo $\delta \in (0,1)$ zgornja meja za verjetnost za napako in naj bo $\alpha := 1-\delta$ stopnja zaupanja.

Izrek 2 (Multivariatna delta metoda)**.** *Naj bo $\mu := \mathbb{E}[Y_n]$ in naj bo $Y_n = (Y_{n1}, Y_{n2}, \ldots, Y_{nk})$ zaporedje naključnih vektorjev, da je $\sqrt{n}(Y_n - \mu) \rightsquigarrow \mathcal{N}(0, \Sigma)$, pri čemer je Σ kovariančna matrika. Naj bo $g : \mathbb{R}^k \to \mathbb{R}$ in naj bo*

$$\nabla g(\mathbf{y}) = \begin{pmatrix} \partial g(\mathbf{y})/\partial y_1 \\ \vdots \\ \partial g(\mathbf{y})/\partial y_k \end{pmatrix}.$$

Nadalje naj bo ∇_μ vrednost $\nabla g(\mathbf{y})$ pri $\mathbf{y} = \mu$ in recimo, da so elementi ∇_μ neničelni. Potem velja

$$\sqrt{n}(g(Y_n) - g(\mu)) \rightsquigarrow \mathcal{N}\left(0, \nabla_\mu^T \Sigma \nabla_\mu\right). \tag{3.7}$$

Za več o multivariatni delta metodi glej [Wasserman, 2006, razdelek 6.5].

Preden navedemo trditev 1 izpostavimo, da je $G(A)$ funkcija $c + dc$ spremenljivk za d-vrednostni atribut, pri čemer je c število razredov pri klasifikaciji.

Trditev 1 ([Jin and Agrawal, 2003])**.** *Naj bo n velikost vzorca in naj bo $G(A) = H(E) - \sum_{i=1}^{d} p_i H(E_i)$ informacijski dobitek d-vrednostnega atributa A. Potem velja*

$$\overline{G}(A) \rightsquigarrow \mathcal{N}\left(G(A), \tau_A^2/n\right), \tag{3.8}$$

pri čemer, če definiramo I kot indeksno množico spremenljivk funkcije G, je

$$\tau_A^2 = \sum_{i \in I} \left(\frac{\partial G}{\partial p_i}\right)^2 p_i(1-p_i).$$

Dokaz. Gre za direktno posledico izreka 2 ob upoštevanju, da so parcialni odvodi G zvezni. □

Naj bosta A_1 in A_2 atributa in naj bosta $\overline{G}(A_1)$ in $\overline{G}(A_2)$ njuni vzorčni oceni. Ker sta to neodvisni naključni spremenljivki, velja

$$\mathsf{Cov}\left(\overline{G}(A_i), \overline{G}(A_j)\right) = \begin{cases} 0, & i \neq j, \\ \mathbb{V}[X_i], & i = j, \end{cases}$$

zato je

$$\overline{G}(A_1) - \overline{G}(A_2) \rightsquigarrow \mathcal{N}\left(G(A_1) - G(A_2), (\tau_1^2 + \tau_2^2)/n\right),$$

saj je kovariančna matrika $\Sigma_{ij} = 0$ za $i \neq j$.

Trditev 2 ([Jin and Agrawal, 2003])**.** *Naj bo A_1 in A_2 različna atributa in naj bo*

$$\epsilon = \frac{z_\alpha}{\sqrt{n}}\sqrt{\tau_1^2 + \tau_2^2}, \tag{3.9}$$

pri čemer z_α označuje $(1-\alpha)$-ti percentil standardne normalne porazdelitve[2]*. Če je $\overline{G}(A_1) - \overline{G}(A_2) \geq \epsilon$, potem z verjetnostjo vsaj α velja $G(A_1) \geq G(A_2)$. Podobno, če je $\overline{G}(A_1) - \overline{G}(A_2) \leq -\epsilon$, potem z verjetnostjo vsaj α velja $G(A_2) \geq G(A_1)$.*

3.4 Primerjava testov

Hoeffdingov test je bolj splošen kot normalni test, ki izkorišča lastnosti informacijskega dobitka in Gini indeksa. Kakšna je velikost vzorca pri fiksnih parametrih za normalni test in za Hoeffdingov test? Iz (3.6) izrazimo

$$n_1 = \left\lceil \frac{R^2 \log(1/\delta)}{2\epsilon^2} \right\rceil \tag{3.10}$$

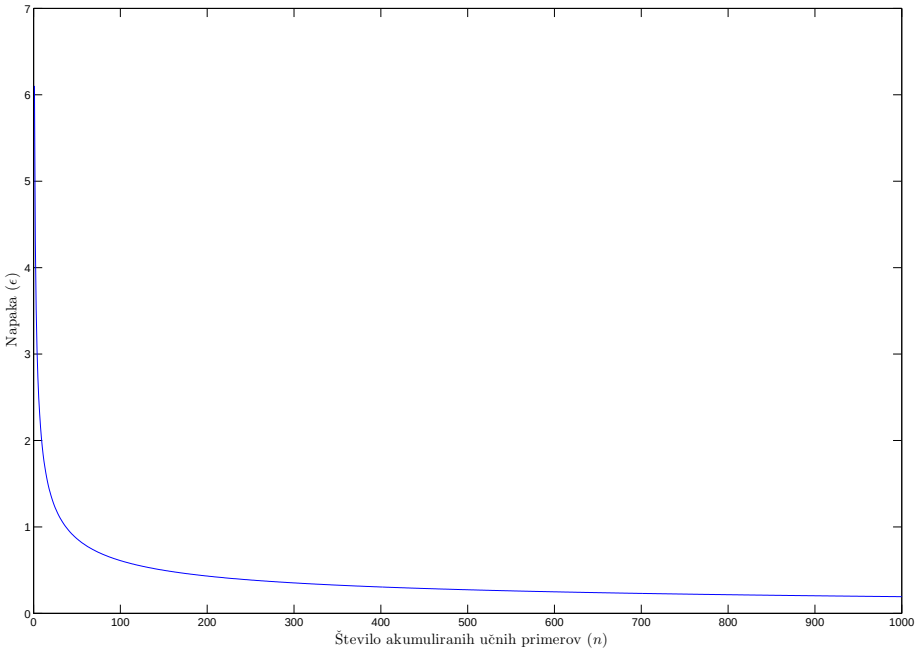
in podobno iz (3.9) izrazimo

$$n_2 = \frac{z_\alpha^2}{\epsilon^2}\sqrt{\tau_1^2 + \tau_2^2}.$$

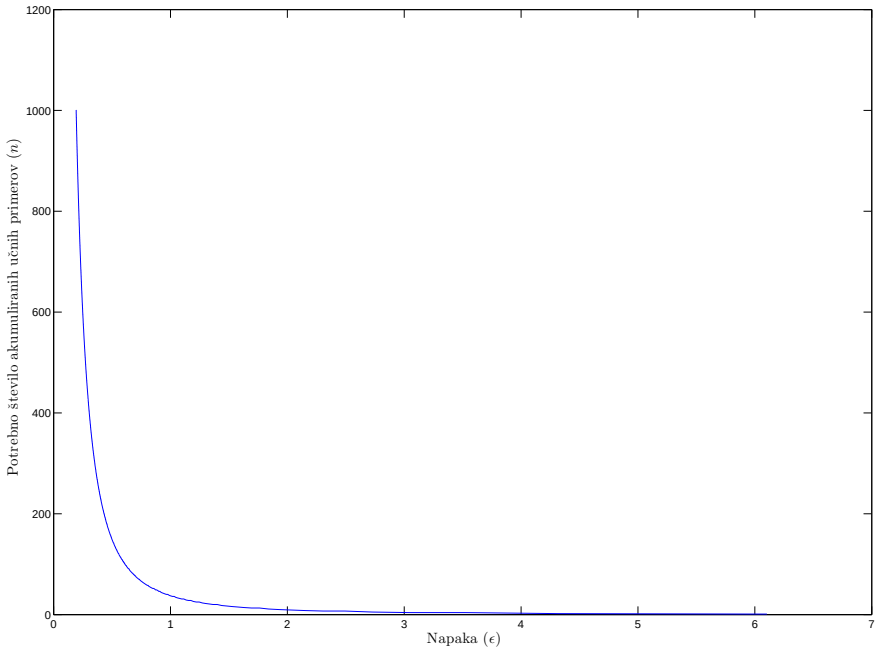
Hitro vidimo, da velja $n_2 \leq n_1$, torej je Hoeffdingov test res bolj konzervativen od Normalnega.

Obnašanje funkcije (3.6) prikazuje slika 3.1.

[2]Naj bo $Z \sim \mathcal{N}(0,1)$ standardno normalno porazdeljena in naj bo $\alpha \in (0,1)$. Potem definiramo z_α kot število, za katerega je $\mathbb{P}(Z > z_\alpha) = \alpha$. (To je inverz gostote verjetnosti standardne normalne porazdelitve v dani točki: $z_\alpha := \Phi^{-1}(1-\alpha)$.)



(a) Obnašanje funkcije (3.6) za $\delta = 10^{-6}$ in $R = \log_2 5$ za $1 \leq n \leq 1000$.



(b) Obnašanje funkcije (3.10) za $\delta = 10^{-6}$ in $R = \log_2 5$ za $0.1 \leq \epsilon \leq 6$.

Slika 3.1: Primer obnašanja Hoeffdingovega testa:

4 Hoeffdingova drevesa

V tem poglavju navedemo splošno metodo za skaliranje razreda učnih algoritmov za učenje na podatkovnih tokovih in nato navedemo varianti dveh najbolj znanih algoritmov za sprotno učenje klasifikacijskih dreves na spremenljivih podatkovnih tokovih. Skiciramo tudi glavne ideje za algoritmi za sprotno učenje regresijskih dreves. Za enega od algoritmov dokažemo, da se s poljubno veliko verjetnostjo nauči odločitvenega drevesa, ki skoraj vedno uporabi enako zaporedje testov, kot bi ga uporabilo paketno-naučeno drevo na celem podatkovnem toku.

4.1 Splošna metoda za učenje na podatkovnih tokovih

Domingos in Hulten v člankih *A general method for scaling up machine learning algorithms and its applications to clustering* [Domingos and Hulten, 2001a], *A general framework for mining massive data streams* [Domingos and Hulten, 2001b] in *Learning from infinite data in finite time* [Domingos and Hulten, 2003] navedeta splošno metodo za skaliranje klasičnih učencev za učenje na podatkovnih tokovih. Tukaj skiciramo glavno idejo njunega pristopa.

Naj bo A učni algoritem in naj bodo $s_1, s_2, \ldots$ koraki tega algoritma. Nadalje naj bo S učna množica N neodvisno in enako porazdeljenih učnih primerov, naj bo $n_i \leq N$ število učnih primerov, uporabljenih v koraku s_i, in naj bo $\mathbf{n} := \langle n_1, n_2, \ldots \rangle$.

V grobem postopek za skaliranje učenca razdelimo na tri korake:

1. izpelji zgornjo mejo za časovno zahtevnost učnega algoritma kot funkcijo velikosti vzorca v vsakem koraku,
2. izpelji zgornjo mejo za relativno izgubo med modelom na končno primerih in modelom na neskončno primerih kot funkcijo velikosti vzorca v vsakem koraku algoritma na končno primerih,
3. minimiziraj časovno mejo (prek velikosti vzorca v vsakem koraku) glede na uporabniško definirane zahteve glede izgube.

Če se algoritem v koraku s_i odloča med končnim številom izbir, naj bo $\delta_i \in (0, 1)$ verjetnost, da algoritem v tem koraku naredi napačno odločitev; če algoritem v koraku s_i ocenjuje realno vrednost, potem naj bo $\delta_i \in (0, 1)$ verjetnost, da se resnična vrednost od ocenjene razlikuje za več kot $\epsilon_i > 0$.

Na podlagi verjetnostne neenakosti — v našem primeru Hoeffdingove neenakosti — izrazi ϵ_i in δ_i kot funkciji n_i. Naj bo $L(M_1, M_2)$ funkcija izgube, ki kaznuje razliko med modeloma M_1 in M_2. Nadalje naj bo M_∞ model, ki se ga nauči A, če v vsakem koraku uporabi neomejeno primerov in naj bo $M_\mathbf{n}$ model, ki se ga nauči A, če v vsakem od korakov s_i uporabi n_i primerov.

Sedaj omejimo izgubo $L(M_\mathbf{n}, M_\infty)$ kot funkcijo ϵ_i in δ_i, in zato kot funkcijo n_i, za vsak korak s_i algoritma A: $\mathbb{P}(L(M_\mathbf{n}, M_\infty) > G_{A,S}(\mathbf{n})) \leq F_{A,S}(\mathbf{n})$. Tako z verjetnostjo vsaj $1 - F_{A,S}(\mathbf{n})$ velja $L(M_\mathbf{n}, M_\infty) \leq G_{A,S}(\mathbf{n})$, pri čemer sta funkciji G in F odvisni od učenca A in učne množice S.

Zgornjo idejo lahko ilustriramo na preprostem primeru učenja pričakovanja naključne spremenljivke X. Naj bodo $X_1, X_2, \ldots, X_n$ neodvisne meritve omejene naključne spremenljivke $a \leq X \leq b$. Po Hoeffdingovi neenakosti je

$$\mathbb{P}\left(S - \mathbb{E}[S] \geq n\epsilon\right) \leq \exp\left(-2n\epsilon^2/R^2\right),$$

pri čemer je $S := X_1 + X_2 + \ldots + X_n$ in $R := b - a$. Naj bo $\delta \in (0, 1)$ zgornja meja za verjetnost za napako. Potem rešujemo $\delta \leq \exp\left(-2n\epsilon^2/R^2\right)$, od koder sledi, da z verjetnostjo vsaj $1 - \delta$ naredimo napako kvečjemu ϵ, pri čemer je

$$\epsilon \geq \sqrt{\frac{R^2 \log(1/\delta)}{2n}}.$$

To pomeni, da se z verjetnostjo vsaj $1-\delta$, za izbran δ, naučimo pravega pričakovanja znotraj odprtega intervala $(\mathbb{E}[X] - \epsilon, \mathbb{E}[X] + \epsilon)$ po

$$n = \left\lceil \frac{R^2 \log(1/\delta)}{2\epsilon^2} \right\rceil$$

meritvah naključne spremenljivke.

Domingos in Hulten v omenjenih člankih metodo uporabita za skaliranje odločitvenih dreves (VFDT), k-means razvrščanja (VFKM), Bayesovih mrež (VFBN) in EM algoritma (VFEM).

4.2 Inkrementalno učenje klasifikacijskih dreves na stacionarnih podatkovnih tokovih

V tem razdelku izpeljemo algoritem za inkrementalno učenje odločitvenih dreves na podatkovnih tokovih, kjer je vir, ki generira podatke, stacionaren.

Učenje odločitvenih dreves je metoda za aproksimacijo hipoteze s funkcijo, ki jo predstavimo z odločitvenim drevesom. Klasični algoritmi za gradnjo odločitvenih dreves izberejo atribut v korenu, ki sam najbolje klasificira učne primere v korenu; koren nato dobi sina za vsako od vrednosti izbranega atributa in učne primere razbije med sinove glede na vrednost izbranega atributa v vsakem od primerov; postopek rekurzivno ponovimo na vsakem od sinov, dokler ne porabimo vseh atributov, ne zmanjka primerov, ali ne dosežemo dovolj dobre napovedne točnosti. Za izbiro atributov se skoraj vedno uporablja mera za ocenjevanje atributov[1], najpogosteje informacijski dobitek ali Gini indeks in njune variante.

Klasični učenci predpostavljajo, da imajo na voljo vse učne primere in da le-ti gredo v delovni pomnilnik. V primeru podatkovnih tokov je primerov za vse praktične namene neskončno, nad vrstnim redom in hitrostjo prihajanja primerov pa nimamo kontrole. Iz prejšnjega poglavja vemo, da lahko oceno atributa ocenimo že iz končne, relativno majhne, podmnožice primerov iz podatkovnega toka. Torej lahko sledimo klasičnemu postopku in v danem listu akumuliramo učne primere, dokler ne velja $\overline{G}(X_a) - \overline{G}(X_b) > \epsilon$, pri čemer sta $\overline{G}(X_a)$ in $\overline{G}(X_b)$ vzorčni oceni ocen najboljših dveh atributov. (Izpostavimo, da je ϵ monotono padajoča funkcija števila učnih primerov.) Kadar je pogoj izpolnjen, razcepimo list na atributu X_a in vse nadaljnje primere podamo ustreznemu sinu. Tako pridemo do algoritma 1. (Skrita predpostavka je, da imajo ostali atributi dovolj majhne dobitke, da lahko verjetnost, da je v resnici najboljši eden od teh atributov, zanemarimo.)

V resnici je dovolj, da v listu hranimo zadostno statistiko primera — to je informacija, ki jo rabimo za izračun hevrističnih ocen — ki je v tem primeru število primerov iz k-tega razreda, za katere i-ti atribut zavzame j-to vrednost. To nam zadošča za izračun ocen verjetnosti, ki jih uporablja večina mer za ocenjevanje atributov, med njimi informacijski dobitek in Gini indeks.

Kadar sta dva atributa skoraj enako dobra, bo algoritem akumuliral primere v nedogled. V ta namen vpeljemo uporabniško definirano konstanto τ, tipično $\tau := 0.05$, ki pove, kdaj dva atributa smatramo za enako dobra in razcepimo list z enim od njiju.

V prejšnjem poglavju smo rekli, da je R zaloga vrednosti (angl. range) naključne spremenljivke. Za klasifikacijska drevesa, ki ocenjujejo atribute z informacijskim dobitkom, je $R := \log_2 C$, pri čemer je C število razredov, ker entropija C-vrednostne diskretne naključne spremenljivke leži na intervalu $[0, \log_2 C]$. (Dokaz: Če so vse vrednosti enako verjetne, potem je entropija $-k/k \cdot \log_2(1/k) = \log_2 k$; če za neko vrednost velja $p_i = 1$, potem je entropija očitno 0; entropija res zavzame minimum in maksimum v teh točkah, ker je mera nečistoče.)

[1] Za problem učenja odločitvenih dreves obstaja vrsta pesimističnih rezultatov iz računske zahtevnosti [Hyafil and Rivest, 1976, Hancock et al., 1996, Laber and Nogueira, 2004, Sieling, 2008, Adler and Heeringa, 2008].

Algoritem 1 Varianta VFDT algoritma za inkrementalno učenje klasifikacijskih dreves.

Vhod: Algoritem vzame podatkovni tok S in parametre n_m, δ in τ.
Izhod: Algoritem lahko v vsakem trenutku vrne odločitveno drevo T in nadaljuje z učenjem.

```
1: Naj bo HT koren drevesa
2: for x ∈ S do // Za vsak učni primer podatkovnega toka
3:     Sortiraj primer v list ℓ z HT
4:     Posodobi zadostno statistiko lista ℓ
5:     Povečaj n_ℓ := n_ℓ + 1
6:     if n_ℓ mod n_m = 0 in primeri v listu ℓ ne pripadajo istemu razredu then
7:         Izračunaj G̅_ℓ(X_i) za vsak atribut
8:         Naj bo X_a najboljši atribut
9:         Naj bo X_b drugi najboljši atribut
10:        Izračunaj ε := √(R² log(1/δ) / (2n_ℓ))
11:        Naj bo ΔG̅ := G̅_ℓ(X_a) − G̅_ℓ(X_b)
12:        if X_a ≠ X_0 in (ΔG̅ > ε ali ΔG̅ ≤ ε < τ) then // Hoeffdingov test
13:            Zamenjaj list ℓ z vozliščem s testom na atributu X_a
14:            for all vrednosti atributa X_a do // Razcepi
15:                Dodaj nov list in inicializiraj njegovo zadostno statistiko
```

Podobno Gini indeks C-vrednostne diskretne naključne spremenljivke leži na intervalu $[0, 1-1/C]$, zato je za Gini indeks $R := 1 - 1/C$. (Dokaz: Gini indeks je največji kadar so vse vrednosti enako verjetne, kar nam da $1 - C/C^2 = 1 - 1/C$, in najmanjši, kadar je verjetnost $p_i = 1$, kar nam da 0; Gini indeks res zavzame minimum in maksimum v teh točkah, ker je mera nečistoče.)

4.2.1 Lastnosti VDFT algoritma

V tem podrazdelku navedemo teoretične rezultate, ki nam zagotavljajo, da se na podatkovnem toku z algoritmom 1 s poljubno veliko verjetnostjo naučimo odločitveno drevo, ki bo za klasifikacijo primerov uporabljajo enaka zaporedja testov kot paketno-naučeno drevo na celem podatkovnem toku.

Izpostavimo, da spodnji rezultati *ne veljajo*[2] za algoritem 1, ker vrstica 1.12 ni le Hoeffdingov test, ampak je pogoj resničen tudi kadar sta najboljša dva atributa praktično enako dobra — kadar za nek uporabniško definiran $\tau > 0$ velja

$$\overline{G}(X_a) - \overline{G}(X_b) \leq \epsilon < \tau, \tag{4.1}$$

kjer tipično postavimo $\tau := 0.05$. Osnovna varianta VFDT algoritma pogoja iz enačbe (4.1) ne uporablja.

Omenimo še, da lahko definiramo patološke klasifikatorje v listih drevesa za katere rezultati tega podrazdelka odpovejo. Algoritem VFDT iz [Domingos and Hulten, 2000] v listih drevesa uporablja večinski klasifikator.

Definicija 1. *Naj bosta T_1 in T_2 odločitveni drevesi in naj bo S podatkovni tok. Potem definiramo*

$$\Delta_e(T_1, T_2) := \sum_{x \in S} \mathbb{P}(x)\, \mathbb{I}[T_1(x) \neq T_2(x)]$$

kot verjetnost, da bosta drevesi dali različno klasifikacijo primera. Podobno definiramo

$$\Delta_i(T_1, T_2) := \sum_{x \in S} \mathbb{P}(x)\, \mathbb{I}[\mathrm{Pot}_1(x) \neq \mathrm{Pot}_2(x)]$$

kot verjetnost, da bosta drevesi uporabili različni poti pri klasifikaciji primera.

[2] Vseeno veljajo podobni rezultati; glavna ideja je podobna; glej [Hulten et al., 2005] za podrobnosti.

Izpostavimo, da je Δ_i strožja od Δ_e, saj za vsa klasifikacijska drevesa T_1, T_2 velja $\Delta_e(T_1, T_2) \leq \Delta_i(T_1, T_2)$, ker pod predpostavko, da v listih uporabljamo večinski klasifikator, iz $\mathrm{Pot}_1(x) = \mathrm{Pot}_2(x)$ sledi $T_1(x) = T_2(x)$.

Naj bo p_ℓ verjetnost, da primer, ki v odločitvenem drevesu doseže nivo ℓ, pade v list na tem nivoju. V nadaljevanju predpostavimo, da za vse nivoje ℓ velja $p_\ell = p$ za nek p. Naj bo $\mathbb{P}(x)$ verjetnost, da algoritem na vhodu vidi vektor atributov (to je učni primer) x.

Izrek 3 ([Domingos and Hulten, 2000])**.** *Naj bo T_δ odločitveno drevo, ki ga vrne algoritem 1 za izbran $\delta \in (0, 1)$ na podatkovnem toku S na potencialno neskončno primerih in naj bo $T_\star$ odločitveno drevo, ki v vsakem vozlišču uporabi atribut z resničnim največjim G. Tedaj velja*

$$\mathbb{E}\left[\Delta_i(T_\delta, T_\star)\right] \leq \delta/p, \tag{4.2}$$

pri čemer je

$$\mathbb{E}\left[\Delta_i(T_\delta, T_\star)\right] = \sum_{x \in S} \mathbb{P}(x)\, \mathbb{P}\left(\mathrm{Pot}_\delta(x) \neq \mathrm{Pot}_\star(x)\right),$$

kjer je S podatkovni tok.

Dokaz. Naj učni primer x pade v list na nivoju ℓ_δ v T_δ in v list na nivoju $\ell_\star$ v $T_\star$. Nadalje naj bo $\ell := \min(\ell_\delta, \ell_\star)$ in naj bo $N_i^\delta(x)$ vozlišče v T_δ skozi katerega gre x na nivoju i. Podobno naj bo $N_i^\star(x)$ vozlišče v $T_\star$ skozi katerega gre x na nivoju i. Označimo $I_i := \mathbb{I}[\mathrm{Pot}_\delta(x)$ in $\mathrm{Pot}_\star(x)$ se ujemata v prvih i nivojih] in postavimo $I_0 := 1$, kjer interpretiramo 1 kot resnično (angl. true) in 0 kot neresnično (angl. false).

Tedaj velja

$$\mathbb{P}(\mathrm{Pot}_\delta(x) \neq \mathrm{Pot}_\star(x)) = \mathbb{P}\left(\bigvee_{i=1}^{\ell} N_i^\delta(x) \neq N_i^\star(x)\right),$$

kar lahko zapišemo kot

$$\sum_{i=1}^{\ell} \mathbb{P}\left(N_i^\delta(x) \neq N_i^\star(x) | I_{i-1}\right).$$

Po konstrukciji — pri gradnji drevesa smo za izbiro atributa uporabili Hoeffdingov test — za vse $i \geq 1$ velja

$$\mathbb{P}(N_i^\delta(x) \neq N_i^\star(x) \mid I_{i-1}) \leq \delta,$$

od koder dobimo

$$\mathbb{P}\left(\mathrm{Pot}_\delta(x) \neq \mathrm{Pot}_\star(x)\right) \leq \ell\delta.$$

Oglejmo si sedaj $\mathbb{E}\left[\Delta_i(T_\delta, T_\star)\right]$, ki je povprečje verjetnosti (čez vsa zaporedja S), da $T_\star$ in naučeno T_δ uporabita različni zaporedji testov pri klasifikaciji naključnega primera. Naj bo L_i množica učnih primerov, ki padejo v list na i-tem nivoju. Tedaj velja

$$\begin{aligned}
\mathbb{E}\left[\Delta_i(T_\star, T_\delta)\right] &= \sum_{x \in S} \mathbb{P}(x)\, \mathbb{P}(\mathrm{Pot}_\delta(x) \neq \mathrm{Pot}_\star(x)) \\
&= \sum_{i \geq 1} \sum_{x \in L_i} \mathbb{P}(x)\, \mathbb{P}(\mathrm{Pot}_\delta(x) \neq \mathrm{Pot}_\star(x)) \\
&\leq \sum_{i \geq 1} \sum_{x \in L_i} \mathbb{P}(x) i\delta \\
&= p\delta \sum_{i \geq 1} i(1-p)^{i-1} \\
&= \delta/p,
\end{aligned}$$

ker po binomskem izreku za negativne koeficiente [Graham et al., 1994, razdelek 5.4] velja

$$\sum_{k\geq 0}(k+1)x^k = (1-x)^{-2}.$$

Uporabili smo predpostavko, da je verjetnost, da bo x klasificiran v list na nivoju ℓ enaka p za vse nivoje, zato velja $\sum_{x\in L_i}\mathbb{P}(x) = p(1-p)^{i-1}$. □

4.3 Inkrementalno učenje klasifikacijskih dreves na spremenljivih podatkovnih tokovih

V tem razdelku izpeljemo algoritem za inkrementalno učenje odločitvenih dreves na spremenljivih podatkovnih tokovih — tedaj vir, ki generira podatke, ni stacionaren in se spreminja s časom, zato mora algoritem zaznavati spremembe in ustrezno prilagajati model.

Za osnovo vzemimo algoritem 1 in mu dodajmo drseče okno W velikosti $w \in \mathbb{N}$, ki se obnaša kot FIFO vrsta. Ko na vhod dobimo nov učni primer, ga dodamo v drseče okno in, če je okno že polno, "pozabimo" in odstranimo iz okna najstarejši učni primer x. To pomeni, da x sortiramo po drevesu — primer bo očitno šel skozi natanko tista vozlišča, skozi katera je šel pri dodajanju — in ustrezno zmanjšamo števce. Drevo se je medtem, ko je bil x v oknu, lahko spremenilo, kar pomeni, da moramo za pravilno pozabljanje vedeti, katera vozlišča je x spremenil. V ta namen vsakemu vozlišču ob kreiranju priredimo monotono naraščajoč ID. Ko dodamo nov primer v drevo, si zapomnimo največjega izmed IDjev listov, v katere je bil primer sortiran, in ta ID, $\mathrm{id}(x)$, priredimo primeru, preden ga shranimo v okno. Pri pozabljanju primera x vozlišču zmanjšamo števec samo, če je $\mathrm{id}(x)$ manjši ali enak IDju vozlišča. Če je podatkovni tok stacionaren, potem odštevanje statistično nima efekta.

Kadar je podatkovni tok spremenljiv, se lahko zgodi, da v nekaterih vozliščih pogoj iz Hoeffdingovega testa ne bo več veljal za izbran atribut. Zato periodično preiskujemo drevo — tipično na vsakih $n_d := 20\,000$ primerov — in iščemo vozlišča, za katera velja $\overline{G}(X_a) - \overline{G}(X_b) \leq \epsilon$ in $\epsilon > \tau$. V vsakem takem vozlišču T začnemo učiti *alternativno drevo* T' in ga dodamo v množico alternativnih dreves tega vozlišča $\mathrm{Alt}(T) := \mathrm{Alt}(T) \cup \{T'\}$. Ko dodajamo ali pozabljamo primer, ga vedno sortiramo po drevesu in ustrezno spremenimo vozlišča in alternativna drevesa vozlišč, skozi katere gre primer.

Vsako vozlišče periodično preklopi v *samoevalvacijski način delovanja* in žrtvuje $n_e \in \mathbb{N}$ zaporednih primerov, da oceni točnost drevesa, ki ga vpenja in točnosti alternativnih dreves. Naj bo T poddrevo in naj bodo $T'_1, \ldots, T'_n$ alternativna drevesa tega vozlišča. Vozlišče T v samoevalvacijskem načinu akumulira $m \in \mathbb{N}$ zaporednih primerov S in jih uporabi za izračun klasifikacijske točnosti vpetega poddrevesa in alternativnih dreves. Naj bo

$$T'_\star := \operatorname*{arg\,min}_{T'\in\mathrm{Alt}(T)} L(T', S)$$

najbolje ocenjeno alternativno drevo; če velja $L(T'_\star, S) < L(T, S)$, potem algoritem zamenja T z $T'_\star$ in pobriše T in $\mathrm{Alt}(T)\backslash\{T'_\star\}$.

Tako pridemo do algoritma 2. Veliko sliko njegovega delovanja prikazuje slika 4.1, kjer črtkane povezave predstavljajo alternativna poddrevesa. Odločitveno drevo v vsakem trenutku odraža vsebino drsečega okna W.

4.4 Regresijska Hoeffdingova drevesa

Algoritme FIMT, FIRT-DD, FIMT-DD in FIMT-MT za inkrementalno učenje regresijskih Hoeffdingovih dreves na spremenljivih podatkovnih tokovih so predstavili [Ikonomovska and Gama, 2008, Ikonomovska et al., 2011, Ikonomovska, 2012]. Tukaj navedemo le, kako se ti algoritmi — razen FIMT, ki uporablja varianto neenakosti Chernoffa [Angluin and Valiant, 1977] — odločajo za cepitev lista.

Algoritem 2 Varianta CVFDT algoritma za inkrementalno učenje prilagodljivih klasifikacijskih dreves.

Vhod: Algoritem vzame podatkovni tok S in parametre $\delta, \tau \in (0,1)$ in $n_m, w, f \in \mathbb{N}$.
Izhod: Algoritem lahko v vsakem trenutku vrne odločitveno drevo T.

Naj bo T prazen list z $\text{Alt}(T) := \{\}$ in naj bo $\text{id}(T) :=$ NextID()
for $x \in S$ **do**
    $W := W \cup \{x\}$
    if $|W| > w$ **then**
        Odstrani najstarejši element x' iz okna W
        Forget(T, x')
    $\text{id}(x) :=$ Process(T, x) `// Dodeli največjega izmed IDjev listov v katere je padel` x
    if $N \mod f = 0$ **then** CheckValidity(T)
procedure Grow(T,x)
    IncrementCounts(T,x)
    if $n_l \mod n_m = 0$ in primeri v listu T ne pripadajo istemu razredu **then**
        Izračunaj $\overline{G}_\ell(X_i)$ za vsak atribut
        Naj bo $X_a := \arg\max_{X_i} \overline{G}(X_i)$ najboljši atribut
        Naj bo $X_b := \arg\max_{X_i \neq X_a} \overline{G}(X_i)$ drugi najboljši atribut
        Izračunaj $\epsilon := \sqrt{\frac{R^2 \log(1/\delta)}{2n_\ell}}$
        Naj bo $\Delta\overline{G} := \overline{G}_\ell(X_a) - \overline{G}_\ell(X_b)$
        if $X_a \neq X_0$ in $\left(\Delta\overline{G} > \epsilon \text{ ali } \Delta\overline{G} \leq \epsilon < \tau\right)$ **then** `// Hoeffdingov test`
            Zamenjaj T z vozliščem na X_a
            for all vrednosti atributa X_a **do**
                Dodaj vozlišču T nov list T' in ga inicializiraj $\text{Alt}(T') := \{\}$ in $\text{id}(T') :=$ NextID()
function Process(T,x)
    Naj bo $t := 0$ trenutni ID
    while T ni list **do**
        for all $T' \in \text{Alt}(T)$ **do** $t := \max(t,$ Process$(T', x))$
        IncrementCounts(T, x)
        $T :=$ NextNode(T, x)
    Grow(T,x)
    return $\max(t, \text{id}(T))$
procedure CheckValidity(T)
    while T ni list **do**
        for all $T' \in \text{Alt}(T)$ **do** CheckValidity(T')
        Naj bo $\overline{G}(X_n) := \max_{X_i \neq X_a} \overline{G}(X_i)$, če T testira na atributu X_a
        Naj bo $\overline{G}(X_b) := \max_{X_i \neq X_n} \overline{G}(X_i)$
        if $\overline{G}(X_n) - \overline{G}(X_b) \geq 0$ in nobeno od $\text{Alt}(T)$ ne testira na X_n v korenu **then**
            Naj bo $\epsilon := \sqrt{\frac{R^2 \log(1/\delta)}{2n}}$
            if $\overline{G}(X_n) - \overline{G}(X_b) > \epsilon$ ali $(\epsilon < \tau$ in $\overline{G}(X_n) - \overline{G}(X_b) \geq \tau/2)$ **then**
                Naj bo T_n novo vozlišče z $\text{id}(T_n) :=$ NextID() in $\text{Alt}(T_n) := \{\}$
                for all vrednosti atributa X_n **do**
                    Dodaj vozlišču T_n nov list T_i in postavi $\text{id}(T_i) :=$ NextID() in $\text{Alt}(T_i) := \{\}$
                Dodaj T_n v množico alternativnih dreves $\text{Alt}(T) := \text{Alt}(T) \cup \{T_n\}$ vozlišča T
`// Se nadaljuje na naslednji strani`

```
42: procedure FORGET(T, x)
43:     while T ni list in id(T) ≤ id(x) do
44:         for all T' ∈ Alt(T) do FORGET(T', x)
45:         DECREMENTCOUNTS(T, x)
46:         T := NEXTNODE(T, x)
```

Recimo, da smo v danem listu drevesa akumulirali n učnih primerov in recimo, da hipotetični binarni test h_A na atributu A razbije množico S na podmnožici S_L in S_R velikosti n_L in n_R. Potem definiramo

$$\mathrm{sdr}(h_A) := \sigma(S) - \frac{n_L}{n}\sigma(S_L) - \frac{n_R}{n}\sigma(S_R),$$

pri čemer je

$$\sigma(S) := \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y})^2}$$

standardni odklon.

Naj bo A najboljši atribut in naj bo B drugi najboljši atribut. Potem je razmerje $r := \mathrm{sdr}(h_B)/\,\mathrm{sdr}(h_A)$ naključna spremenljivka in očitno je $r \in [0, 1]$. Nadalje naj bodo $r_1, r_2, \ldots, r_n$ razmerja med najboljšima atributoma za zaporednih n primerov. Potem po Hoeffdingovi neenakosti velja

$$\mathbb{P}\left(r \in [\overline{r} - \epsilon, \overline{r} + \epsilon]\right) \geq 1 - \delta,$$

pri čemer je $\overline{r} := (r_1 + r_2 + \ldots + r_n)/n$ vzorčno povprečje, $\delta \in (0, 1)$, in

$$\epsilon := \sqrt{\frac{\log(1/\delta)}{2n}}.$$

Naj bo S_A zmanjšanje variance po testu na atributu A in naj bo S_B zmanjšanje variance po testu na atributu B. Potem razcepimo na atributu A, če velja

$$S_B/S_A < 1 - \epsilon, \tag{4.3}$$

pri čemer je n število akumuliranih učnih primerov. Bolj splošno, recimo, da imamo d-vrednostni atribut A. Potem je

$$\mathrm{sdr}(h_A) = \sigma(S) - \sum_{i=1}^{d}\frac{n_i}{n}\sigma(S_i).$$

(Ker je zaloga vrednosti (angl. range) naključne spremenljivke r zaprt enotski interval $[0, 1]$, je $R = 1$, zato je enačba za ϵ poenostavljena.)

Torej ideja je, da kadar velja $S_B/S_A < 1 - \epsilon$, je A resnično najboljši atribut z verjetnostjo vsaj $1 - \delta$, ker sta A in B "dovolj narazen".

V literaturi nismo zasledili teoretičnih rezultatov za FIMT, zato postavimo naslednje vprašanje.

Vprašanje 1. *Naj bo $\epsilon > 0$ in naj bosta T_1 in T_2 regresijski drevesi. Definirajmo*

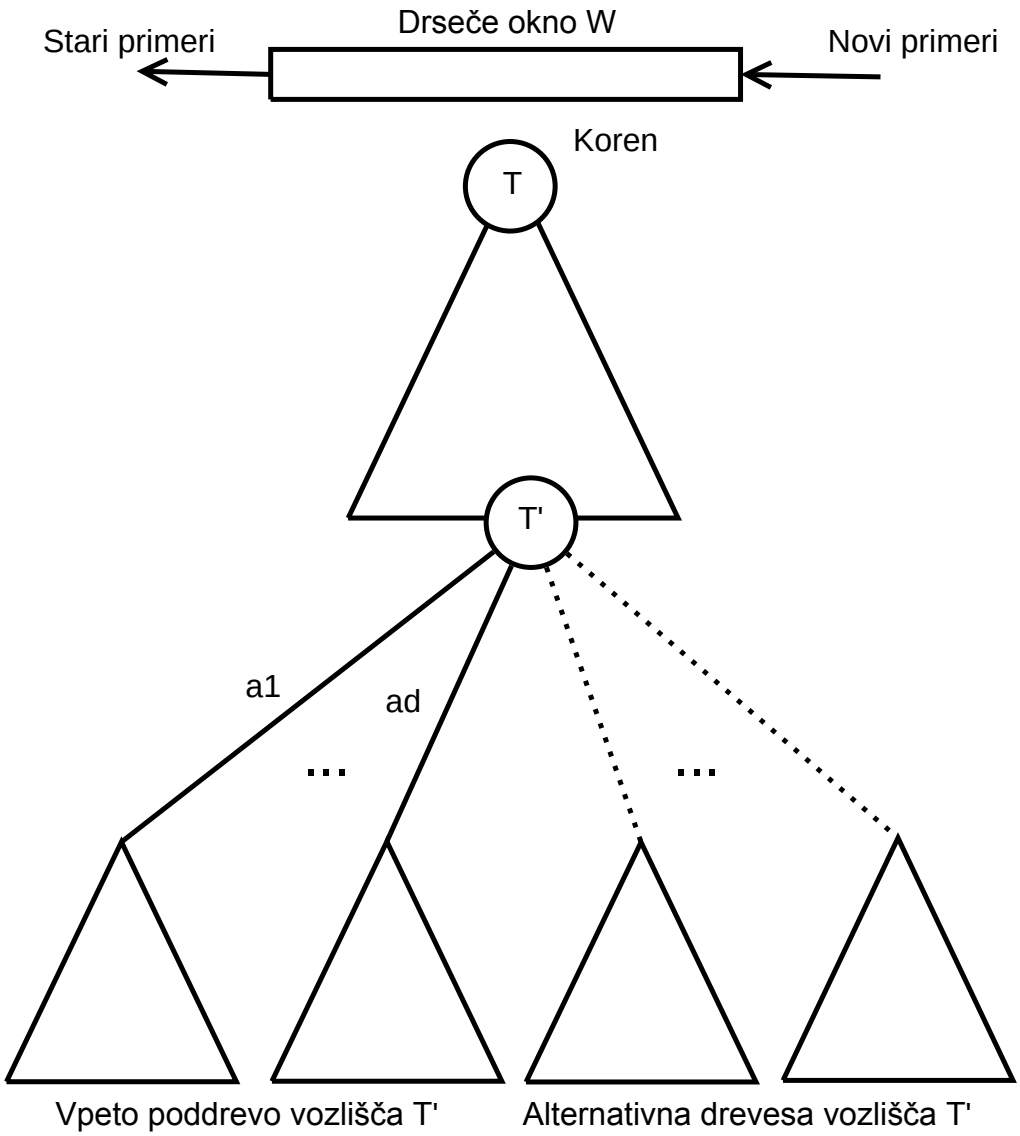
$$\Delta_e^{\epsilon}(T_1, T_2) := \sum_{x \in S}\mathbb{P}(x)\,\mathbb{I}[|T_1(x) - T_2(x)| > \epsilon] \tag{4.4}$$

kot verjetnost, da se bosta napovedi razlikovali za več kot ϵ. Ali za FIMT *lahko dokažemo podobne rezultate kot [Domingos and Hulten, 2000] za* VFDT*?*

Algoritem 3 Grob opis FIMT-DD algoritma za inkrementalno učenje regresijskih dreves.

Vhod: Algoritem vzame podatkovni tok S.
Izhod: V vsakem trenutku lahko vrne regresijsko drevo.

```
1: for x ∈ S do
2:     Naj bo T' := TRAVERSE(T, x) list
3:     Posodobi zadostno statistiko lista T'
4:     Posodobi uteži perceptrona v tem listu
5:     if n mod n_m = 0 then
6:         Izračunaj najboljši razcep za vsak atribut
7:         Naj bosta S_a in S_b najboljši in drugi najboljši atribut
8:         if S_b/S_a < 1 − sqrt(log(1/δ)/(2n)) then
9:             Razcepi list
```



Slika 4.1: Delovanje algoritma 2 v splošnem: odločitveno drevo v vsakem trenutku ustreza vsebini drsečega okna W. Drevo akumulira nove primere in pozablja stare s spreminjanjem zadostne statistike; na podlage le-te prilagaja model vsebini drsečega okna.

5 Inkrementalne formule in algoritmi za entropijo in Gini indeks

V tem poglavju navedemo preproste inkrementalne formule in algoritme za informacijsko-teoretično entropijo [Shannon, 1948] in Gini indeks — popularni meri za ocenjevanje atributov pri učenju odločitvenih dreves in odločitvenih pravil — podobne tistim za pričakovanje in varianco [Chan et al., 1979]. Presenetljivo je, da kljub vse večjemu zanimanju za rudarjenje podatkovnih tokov v literaturi nismo zasledili podobnih rezultatov. (Rezultati tega poglavja so povzeti v [Sovdat, 2013].)

5.1 Inkrementalne formule za Gini indeks

V tem razdelku dokažemo preproste inkrementalne formule za računanje Gini indeksa. Te formule uporabimo z drsečimi okni in bledečimi faktorji; tako dobimo inkrementalne algoritme za računanje Gini indeksa na spremenljivih podatkovnih tokovih.

Definicija 2. *Naj bo $\{x_i\}_{i=1}^n$ vzorec pozitivnih realnih števil, naj bo S_n vsota vzorčnih elementov in naj bo $p_i := x_i/S_n$. Potem definiriamo Gini indeks vzorca kot*

$$G_n := 1 - \sum_{i=1}^{n} p_i^2 = 1 - \sum_{i=1}^{n} \left(\frac{x_i}{S_n}\right)^2 = 1 - \frac{1}{S_n^2}\sum_{i=1}^{n} x_i^2.$$

Preden nadaljujemo zapišemo naslednjo očitno, a izredno uporabno, enakost kot lemo.

Lema 2. *Naj bo $\{x_i\}_{i=1}^n$ vzorec pozitivnih realnih števil, naj bo S_n vsota vzorčnih elementov in naj bo G_n vzorčni Gini indeks. Tedaj velja*

$$\sum_{i=1}^{n} x_i^2 = S_n^2(1 - G_n). \tag{5.1}$$

Dokaz. Razpišemo desno stran enačbe in računamo:

$$S_n^2(1 - G_n) = S_n^2\left(1 + \frac{1}{S_n^2}\sum_{i=1}^{n} x_i^2 - 1\right) = \sum_{i=1}^{n} x_i^2.$$

□

Naslednja trditev nam da posodobitveno formulo, če se eden od vzorčnih elementov poveča za 1.

Trditev 3. *Naj bo $\{x_i\}_{i=1}^n$ vzorec pozitivnih realnih števil in naj bo G_n vzorčni Gini indeks. Recimo, da x_i postane $x_i + 1$. Naj bo G'_n Gini indeks spremenjenega vzorca. Tedaj velja*

$$G'_n = 1 - \frac{1}{(S_n + 1)^2}\left(S_n^2\,(1 - G_n) + 2x_i + 1\right). \tag{5.2}$$

Dokaz. Očitno velja

$$\begin{aligned} G'_n &= 1 - \frac{1}{S'^2_n} \sum_{i=1}^{n} x'^2_i \\ &= 1 - \frac{1}{(S_n+1)^2} \left(2x_i + 1 + \sum_{i=1}^{n} x_i^2 \right) \\ &= 1 - \frac{1}{(S_n+1)^2} \left(S_n^2 (1 - G_n) + 2x_i + 1 \right), \end{aligned}$$

pri čemer zadnja enakost sledi po lemi 2. □

Naslednji izrek posploši trditev 3.

Izrek 4. *Naj bo $\{x_i\}_{i=1}^n$ vzorec pozitivnih realnih števil in naj bo G_n vzorčni Gini indeks. Recimo, da se elementi x_i povečajo za $r_i > 0$ za $i \in I$, pri čemer je I indeksna množica. Nadalje naj bo $r := r_1 + r_2 + \ldots + r_n$, pri čemer postavimo $r_i := 0$ za $i \notin I$. Naj bo G'_n Gini indeks spremenjenega vzorca. Tedaj velja*

$$G'_n = 1 - \frac{1}{(S_n+r)^2} \left(S_n^2 (1 - G_n) + \sum_{i \in I} \left(2x_i r_i + r_i^2 \right) \right). \tag{5.3}$$

Dokaz. Očitno velja

$$\begin{aligned} G'_n &= 1 - \frac{1}{(S_n+r)^2} \sum_{i=1}^{n} (x_i + r_i)^2 \\ &= 1 - \frac{1}{(S_n+r)^2} \sum_{i=1}^{n} \left(x_i^2 + 2x_i r_i + r_i^2 \right) \\ &= 1 - \frac{1}{(S_n+r)^2} \left(\sum_{i=1}^{n} x_i^2 + \sum_{i \in I} \left(2x_i r_i + r_i^2 \right) \right) \\ &= 1 - \frac{1}{(S_n+r)^2} \left(S_n^2 (1 - G_n) + \sum_{i \in I} \left(2x_i r_i + r_i^2 \right) \right), \end{aligned}$$

pri čemer zadnja enakost sledi po lemi 2. □

Izpostavimo, da če se poveča k števcev, enačba (5.3) zahteva le $O(k)$ operacij.

Naslednja trditev nam da posodobitveno formulo za Gini indeks, kadar v vzorec pride novo pozitivno realno število.

Trditev 4. *Naj bo $\{x_i\}_{i=1}^n$ vzorec pozitivnih realnih števil, naj bo S_n vsota vzorčnih elementov in naj bo G_n vzorčni Gini indeks. Recimo, da pride v vzorec nov element $x_{n+1} > 0$. Naj bo G_{n+1} Gini indeks novega vzorca. Tedaj velja*

$$G_{n+1} = 1 - \frac{1}{(S_n + x_{n+1})^2} \left(S_n^2 (1 - G_n) + x_{n+1}^2 \right). \tag{5.4}$$

Dokaz. Očitno je

$$\begin{aligned} G_{n+1} &= 1 - \frac{1}{(S_n + x_{n+1})^2} \left(\sum_{i=1}^{n} x_i^2 + x_{n+1}^2 \right) \\ &= 1 - \frac{1}{(S_n + x_{n+1})^2} \left(S_n^2 (1 - G_n) + x_{n+1}^2 \right), \end{aligned}$$

pri čemer zadnja enakost velja zaradi leme 2. □

Naslednji izrek posploši trditev 4 in da posodobitveno formulo za Gini indeks kadar "staknemo" dva vzorca.

Izrek 5. *Naj bosta $\{x_i\}_{i=1}^m$ in $\{y_i\}_{i=1}^n$ vzorca pozitivnih realnih števil, naj bosta R_m in S_n vsoti vzorčnih elementov, in naj bosta F_m in G_n vzorčna Gini indeksa. Nadalje naj bo*

$$z_i := \begin{cases} x_i, & 1 \leq i \leq m, \\ y_{i-m}, & m+1 \leq i \leq n+m. \end{cases}$$

Potem je Gini indeks H_{n+m} vzorca $\{z_i\}_{i=1}^{n+m}$ enak

$$H_{n+m} = 1 - \frac{1}{(R_m + S_n)^2} \left(R_m^2 (1 - F_m) + S_n^2 (1 - G_n) \right).$$

Dokaz. Po definiciji je

$$\begin{aligned} H_{n+m} &= 1 - \frac{1}{(R_m + S_n)^2} \sum_{i=1}^{n+m} z_i^2 \\ &= 1 - \frac{1}{(R_m + S_n)^2} \left(\sum_{i=1}^{m} x_i^2 + \sum_{i=1}^{n} y_i^2 \right) \\ &= 1 - \frac{1}{(R_m + S_n)^2} \left(R_m^2 (1 - F_m) + S_n^2 (1 - G_n) \right), \end{aligned}$$

pri čemer zadnjo enakost dobimo, če dvakrat uporabimo lemo 2. □

Izrek 6. *Naj bosta $\{x_i\}_{i=1}^n$ in $\{y_i\}_{i=1}^n$ vzorca pozitivnih realnih števil, naj bosta R_n in S_n vsoti vzorčnih elementov in naj bosta F_n in G_n vzorčna Gini indeksa. Nadalje naj bo $z_i := x_i + y_i$ za $1 \leq i \leq n$. Potem je Gini indeks G "unije" enak*

$$G = 1 - \frac{1}{(R_n + S_n)^2} \left(R_n^2 \left(1 - F_n\right) + S_n^2 \left(1 - G_n\right) + 2 \sum_{i=1}^{n} x_i y_i \right)$$

Dokaz. Po definiciji velja

$$F_n = 1 - \frac{1}{R_n^2} \sum_{i=1}^{n} x_i^2$$

in

$$G_n = 1 - \frac{1}{S_n^2} \sum_{i=1}^{n} y_i^2.$$

Rezultat je sedaj direktna posledica izreka 4. □

5.1.1 Algoritmi za računanje Gini indeksa na spremenljivih podatkovnih tokovih

Podatkovni tokovi so inherentno spremenljivi in ponavadi nas zanima "aktualni" Gini indeks. V tem razdelku navedemo dva algoritma za izračun aktualnega Gini indeksa — eden (algoritem 4) uporablja drseče okno, drugi (algoritem 5) pa bledeče faktorje.

Algoritem 4 računa Gini indeks zadnjih $w \in \mathbb{N}$ elementov podatkovnega toka. V ta namen uporablja drseče okno velikost w in zato porabi $O(w)$ besed pomnilnika. Tukaj je w parameter, s katerim definiramo, katera podmnožica podatkovnega toka je za nas aktualna.

Algoritem 5 računa aktualni Gini indeks z uporabo bledečih faktorjev — doprinosi posameznih elementov so uteženi z utežmi $\{\alpha^k : k \in \mathbb{N}\}$, za fiksen $\alpha \in (0, 1]$, glede na njihovo starost. Tukaj aktualnost definiramo z ustrezno izbiro bledečega faktorja α. Izpostavimo, da ima ta pristop majhno konstantno prostorsko zahtevnost v primerjavi s prejšnjim.

Algoritem 4 Algoritem za računanje Gini indeksa z uporabo drsečega okna.

Vhod: Velikost drsečega okna $w \in \mathbb{N}$.
Izhod: Gini indeks vsebine drsečega okna v vsakem trenutku.

Naj bo $W := \{\}$ drseče okno
Naj bo $n := 0$ število vseh primerov in naj bo $n_i := 0$ število primerov v i-tem razredu
Naj bo $g := 0$ trenutni Gini indeks
for $x \in S$ **do**
    if $|W| > w$ **then**
        Odstrani najstarejši element x', iz i-tega razreda, iz drsečega okna W
        Posodobi $g := \textsc{Dec}(g, n, n_i)$
    Dodaj $W := W \cup \{x\}$ element iz i-tega razreda
    Posodobi $g := \textsc{Inc}(g, n, n_i)$
function $\textsc{Add}(g,\ n,\ n_i)$
    Posodobi $n := n + n_i$
    return $1 - \frac{1}{n^2}\left((n - n_i)^2(1 - g) + n_i^2\right)$
function $\textsc{Del}(g,\ n,\ n_i)$
    Posodobi $n := n - n_i$
    return $1 - \frac{1}{n^2}\left((n + n_i)^2(1 - g) - n_i^2\right)$
function $\textsc{Inc}(g,\ n,\ n_i)$
    Posodobi $n := n + 1$ in $n_i := n_i + 1$
    return $1 - \frac{1}{n^2}\left((n - 1)^2(1 - g) + 2n_i - 1\right)$
function $\textsc{Dec}(g,\ n,\ n_i)$
    Posodobi $n := n - 1$ in $n_i := n_i - 1$
    return $\frac{1}{n^2}\left((n + 1)^2(1 - g) - 2n_i - 1\right)$

Algoritem 5 Algoritem za računanje Gini indeksa z uporabo faktorjev pozabljanja.

Vhod: Bledeči faktor $\alpha \in (0, 1]$ in podatkovni tok S.
Izhod: Aktualni Gini indeks v vsakem trenutku.

Naj bo $n := 0$ število vseh primerov in naj bo $n_i := 0$ število primerov v i-tem razredu
Naj bo $g := 0$ trenutni Gini indeks
for $x \in S$ **do**
    Posodobi Gini indeks $g := 1 - \frac{1}{(n+1)^2}\left(n^2(1 - \alpha g) + 2n_i + 1\right)$
    Posodobi števce $n := n + 1$ in $n_i := n_i + 1$

5.2 Inkrementalne formule za entropijo

V tem razdelku navedemo analogne rezultate za entropijo.

Definicija 3. *Naj bo $\{x_i\}_{i=1}^n$ vzorec pozitivnih realnih števil in naj bo $S_n := x_1 + x_2 + \ldots + x_n$ vsota vzorčnih elementov. Potem definiramo entropijo H_n tega vzroca kot*

$$H_n := -\sum_{i=1}^{n} \frac{x_i}{S_n} \log_2 \frac{x_i}{S_n}.$$

Najprej dokažemo naslednjo tehnično lemo.

Lema 3. *Naj bo $\{x_i\}_{i=1}^n$ vzorec pozitivnih realnih števil, naj bo S_n vsota vzorčnih elementov in naj bo H_n vzorčna entropija. Tedaj za vsako realno število $R > 0$ velja*

$$-\sum_{i=1}^{n} \frac{x_i}{S_n + R} \log_2 \frac{x_i}{S_n + R} = \frac{S_n}{S_n + R} \left(H_n - \log_2 \frac{S_n}{S_n + R} \right). \tag{5.5}$$

Dokaz. Pišimo $\frac{x_i}{R + S_n} = 1 \cdot \frac{x_i}{R + S_n} = \frac{S_n}{S_n} \frac{x_i}{R + S_n} = \frac{x_i}{S_n} \frac{S_n}{R + S_n}$. Potem očitno velja

$$\begin{aligned}
-\sum_{i=1}^{n} \frac{x_i}{S_n + R} \log_2 \frac{x_i}{S_n + R} &= -\sum_{i=1}^{n} \frac{x_i}{S_n} \frac{S_n}{S_n + R} \log_2 \left(\frac{x_i}{S_n} \frac{S_n}{S_n + R} \right) \\
&= -\sum_{i=1}^{n} \frac{x_i}{S_n} \frac{S_n}{S_n + R} \left(\log_2 \frac{x_i}{S_n} + \log_2 \frac{S_n}{S_n + R} \right) \\
&= -\frac{S_n}{S_n + R} \sum_{i=1}^{n} \frac{x_i}{S_n} \log_2 \frac{x_i}{S_n} - \frac{S_n}{S_n + R} \sum_{i=1}^{n} \frac{x_i}{S_n} \log_2 \frac{S_n}{S_n + R} \\
&= \frac{S_n}{S_n + R} H_n - \frac{S_n}{S_n + R} \log_2 \frac{S_n}{S_n + R} \sum_{i=1}^{n} \frac{x_i}{S_n} \\
&= \frac{S_n}{S_n + R} \left(H_n - \log_2 \frac{S_n}{S_n + R} \right).
\end{aligned}$$

□

Naslednja trditev da preprosto inkrementalno formulo, če v vzorec "pride" novo pozitivno realno število $x_i > 0$.

Trditev 5 ([Schlichting and Sovdat, 2013])**.** *Naj bosta H_n in S_n vzorčna entropija in vsota vzorčnih elementov, in recimo, da v porazdelitev pride novo pozitivno realno število $x_{n+1} > 0$. Tedaj velja*

$$H_{n+1} = \frac{S_n}{S_{n+1}} \left(H_n - \log_2 \frac{S_n}{S_{n+1}} \right) - \frac{x_{n+1}}{S_{n+1}} \log_2 \frac{x_{n+1}}{S_{n+1}}. \tag{5.6}$$

Dokaz. Po definiciji velja

$$\begin{aligned}
H_{n+1} &= -\sum_{i=1}^{n+1} \frac{x_i}{S_{n+1}} \log_2 \frac{x_i}{S_{n+1}} \\
&= -\frac{x_{n+1}}{S_{n+1}} \log_2 \frac{x_{n+1}}{S_{n+1}} - \sum_{i=1}^{n} \frac{x_i}{S_{n+1}} \log_2 \frac{x_i}{S_{n+1}} \\
&= \frac{S_n}{S_{n+1}} \left(H_n - \log_2 \frac{S_n}{S_{n+1}} \right) - \frac{x_{n+1}}{S_{n+1}} \log_2 \frac{x_{n+1}}{S_{n+1}},
\end{aligned}$$

pri čemer zadnja enakost sledi iz leme 3. □

Naslednji izrek je posplošitev zgornje trditve in da formulo za entropijo "stika" dveh vzorcev na podlagi vzorčnih entropij F_m in H_n in vsot vzorčnih elementov R_m in S_n.

Izrek 7. *Naj bosta $\{x_i\}_{i=1}^m$ in $\{y_i\}_{i=1}^n$ vzorca pozitivnih realnih števil in naj bosta $R_m := x_1 + x_2 + \ldots + x_m$ in $S_n := y_1 + y_2 + \ldots + y_n$ vsoti vzorčnih elementov. Nadalje naj bosta F_m in H_n vzorčni entropiji. Naj bo*

$$z_i := \begin{cases} x_i, & 1 \leq i \leq m, \\ y_{i-m}, & m+1 \leq i \leq m+n, \end{cases}$$

in naj bo $Z_{m+n} := z_1 + z_2 + \ldots + z_{m+n} = R_m + S_n$. Tedaj velja

$$H_{m+n} = \frac{R_m}{Z_{m+n}} \left(F_m - \log_2 \frac{R_m}{Z_{m+n}} \right) + \frac{S_n}{Z_{m+n}} \left(H_n - \log_2 \frac{S_n}{Z_{m+n}} \right). \tag{5.7}$$

Dokaz. Podobno kot prej imamo

$$\begin{aligned} -\sum_{i=1}^{m+n} \frac{z_i}{Z_{m+n}} \log_2 \frac{z_i}{Z_{m+n}} &= -\sum_{i=1}^{m} \frac{x_i}{Z_{m+n}} \log_2 \frac{x_i}{Z_{m+n}} - \sum_{i=1}^{n} \frac{y_i}{Z_{m+n}} \log_2 \frac{y_i}{Z_{m+n}} \\ &= \frac{R_m}{Z_{m+n}} \left(F_m - \log_2 \frac{R_m}{Z_{m+n}} \right) + \frac{S_n}{Z_{m+n}} \left(H_n - \log_2 \frac{S_n}{Z_{m+n}} \right), \end{aligned}$$

pri čemer za zadnjo enakost dvakrat uporabimo lemo 3 . □

Trditev 5 je v resnici posledica izreka 7, če enačbo (5.7) uporabimo nad H_n in x_{n+1} in gledamo na x_{n+1} kot na vzorec z enim samim elementom.

Izreku 8 nam da posodobitveno entropijsko formulo, če se nekateri elementi x_i za povečajo za $r_i > 0$ za $i \in I$, pri čemer je I indeksna množica.

Izrek 8. *Naj bo $\{x_i\}_{i=1}^n$ vzorec pozitivnih realnih števil in naj bo S_n vsota vzorčnih elementov. Naj bo H_n entropija vzorca. Recimo, da se x_i poveča za $r_i > 0$ za $i \in I$. Naj bo $r := r_1 + r_2 + \ldots + r_n$, pri čemer postavimo $r_i := 0$ za $i \notin I$. Potem entropija H_n postane*

$$\frac{S_n}{S_n + r} H_n - \frac{S_n}{S_n + r} \log_2 \frac{S_n}{S_n + r} - \sum_{i \in I} \left(\frac{x_i + r_i}{S_n + r} \log_2 \frac{x_i + r_i}{S_n + r} - \frac{x_i}{S_n} \log_2 \frac{x_i}{S_n} \right). \tag{5.8}$$

Dokaz. Ideja je, da gledamo na $\dfrac{x_i + r_i}{S_n + r}$ kot na nove elemente, uporabimo izrek 7 in potem odštejemo "stare" elemente $\dfrac{x_i}{S_n}$. □

Če se spremeni k vzorčnih elementov, potrebujemo le $O(k)$ operacij.

Izpeljani postopki se ne zdijo primerni za numerične aplikacije, ker formule postanejo problematične, kadar je x_n majhen v primerjavi s S_n, čeprav se to v eksperimentih — teh zaradi nepopolnosti in časovne stiske ne vključimo — ni pokazalo.

5.2.1 Algoritmi za računanje entropije na spremenljivih podatkovnih tokovih

V tem podrazdelku navedemo dva algoritma za računanje "aktualne" entropije, podobno kot smo to naredili za Gini indeks — eden (algoritem 6) uporablja drseča okna, drugi (algoritem 7) pa bledeče faktorje.

Algoritem 6, podobno kot algoritem 4, vzame parameter $w \in \mathbb{N}$, ki določa velikost drsečega okna in s tem definira kaj je za nas aktualna entropija. Prostorska zahtevnost je očitno $O(w)$. Izpostavimo, da bi si želeli algoritem, ki bi sam prilagajal velikost drsečega okna, podobno kot ADWIN — zaradi inherentno spremenljive narave podatkovnih tokov tudi "optimalen" parameter w ni fiksen, ampak se spreminja s časom.

Algoritem 7, podobno kot algoritem 5, definira aktualnost z uporabo bledečega faktorja $\alpha \in (0, 1]$, ki je edini parameter algoritma, in ima zato majhno konstantno prostorsko zahtevnost. Tudi tukaj so doprinosi posameznih elementov uteženi z utežmi $\{\alpha^k : k \in \mathbb{N}\}$ glede na starost elementa.

Algoritem 6 Računanje entropije z drsečim oknom.

Vhod: Velikost drsečega okna $w \in \mathbb{N}$.
Izhod: Trenutna entropija.

Naj bo $W := \{\}$ drseče okno in naj bo $h := 0$ trenutna entropija.
Naj bo $n := 0$ število vseh primerov in naj bo $n_i := 0$ število primerov v i-tem razredu.
for $x \in S$ **do** `// Manjka primer uporabe ADD in DEC.`
    if $|W| > w$ **then**
        Odstrani najstarejši element x', z oznako razreda j, iz drsečega okna W
        Posodobi $h := \textsc{Dec}(h, n, n_j)$
    Dodaj $W := W \cup \{x\}$ nov element iz razreda i
    Posodobi $h := \textsc{Inc}(h, n, n_i)$
function $\textsc{Add}(h,\, n,\, n_i)$
    Posodobi $n := n + 1$ in inicializiraj $n_i := 1$
    return $\dfrac{n}{n - n_i}\left(h - \log_2 \dfrac{n}{n - n_i}\right) - \dfrac{n_i}{n} \log_2 \dfrac{n_i}{n}$
function $\textsc{Del}(h,\, n,\, n_i)$
    Posodobi $n := n - n_i$ in ponastavi števec $n_i := 0$ za razred i
    return $\dfrac{n + n_i}{n}\left(h + \dfrac{n_i}{n + n_i} \log_2 \dfrac{n_i}{n + n_i}\right) + \log_2 \dfrac{n}{n + n_i}$
function $\textsc{Inc}(h,\, n,\, n_i)$
    Posodobi $n := n + 1$ in $n_i := n_i + 1$
    return $\dfrac{n - 1}{n}\left(h - \log_2 \dfrac{n - 1}{n}\right) - \dfrac{n_i}{n} \log_2 \dfrac{n_i}{n} + \dfrac{n_i - 1}{n - 1} \log_2 \dfrac{n_i - 1}{n - 1}$
function $\textsc{Dec}(h,\, n,\, n_i)$
    Posodobi $n := n - 1$ in $n_i := n_i - 1$ `// Predpostavljamo` $n_i \geq 1$
    return $\dfrac{n + 1}{n}\left(h + \dfrac{n_i + 1}{n + 1} \log_2 \dfrac{n_i + 1}{n + 1} - \dfrac{n_i}{n} \log_2 \dfrac{n_i}{n}\right) + \log_2 \dfrac{n}{n + 1}$

5.3 Komentar

V tem poglavju smo navedli formule in algoritme za inkrementalno računanje entropije in Gini indeksa, ki sta popularni meri za ocenjevanje atributov v učenju odločitvenih dreves in odločitvenih pravil.

Potencialen problem predstavlja numerična stabilnost formul in algoritmov, ki ga v tem delu nismo naslovili. Drugi nenaslovljen problem je prilagajanje velikosti drsečega okna, kot to delata [Bifet and Gavaldà, 2007] pri algoritmu ADWIN.

Zanimivo bi bilo izpeljati podobne rezultate za ostale mere za ocenjevanje atributov. Glavna motivacija za izpeljavo rezultatov tega poglavja je njihova uporaba v VFDT in ostalih inkrementalnih učnih algoritmih, ki entropijo na novo naračunajo vsakih n_m učnih primerov, pri čemer tipično velja $100 \leq n_m \leq 300$.

V času tega pisanja se zdita zanimivi naslednji dve vprašanji:

- ali lahko karakteriziramo razred vzorčnih statistik, za katere obstajajo "dovolj lepe" inkrementalne formule,
- ali obstajajo scenariji, kjer so navedene formule in algoritmi nujni — drugače povedano, ali lahko problem vedno učinkovito rešimo s ponovnim izračunom vzorčnih statistik, kar naredi navedene rezultate praktično neuporabne.

Algoritem 7 Algoritem za računanje entropije z bledečimi faktorji.

Vhod: Bledeči faktor $\alpha \in (0, 1]$ in podatkovni tok S.
Izhod: Aktualna entropija v vsakem trenutku.

Naj bo $n := 0$ število vseh primerov in naj bo $n_i := 0$ število primerov v i-tem razredu
Naj bo $h := 0$ trenutna entropija
for $x \in S$ **do**
    Naj bo i razred elementa x
    Posodobi entropijo $h := \frac{n}{n+1}\left(\alpha h - \log_2 \frac{n}{n+1}\right) - \frac{n_i+1}{n+1}\log_2 \frac{n_i+1}{n+1}$
    Posodobi števce $n := n + 1$ in $n_i := n_i + 1$

Del II

Implementacija

6 Implementacija algoritma

"I love debugging. It's a source of new and unexpected."

—S.V. Zubkov

V tem poglavju zelo grobo opišemo našo implementacijo inkrementalnega učenja odločitvenih dreves.

6.1 Uvod

V tem razdelku podamo globalno sliko implementacije, ne da bi se spuščali v podrobnosti. V naslednjih razdelkih navedemo algoritme za računanje ocen atributov in diskretizacijo numeričnih atributov.

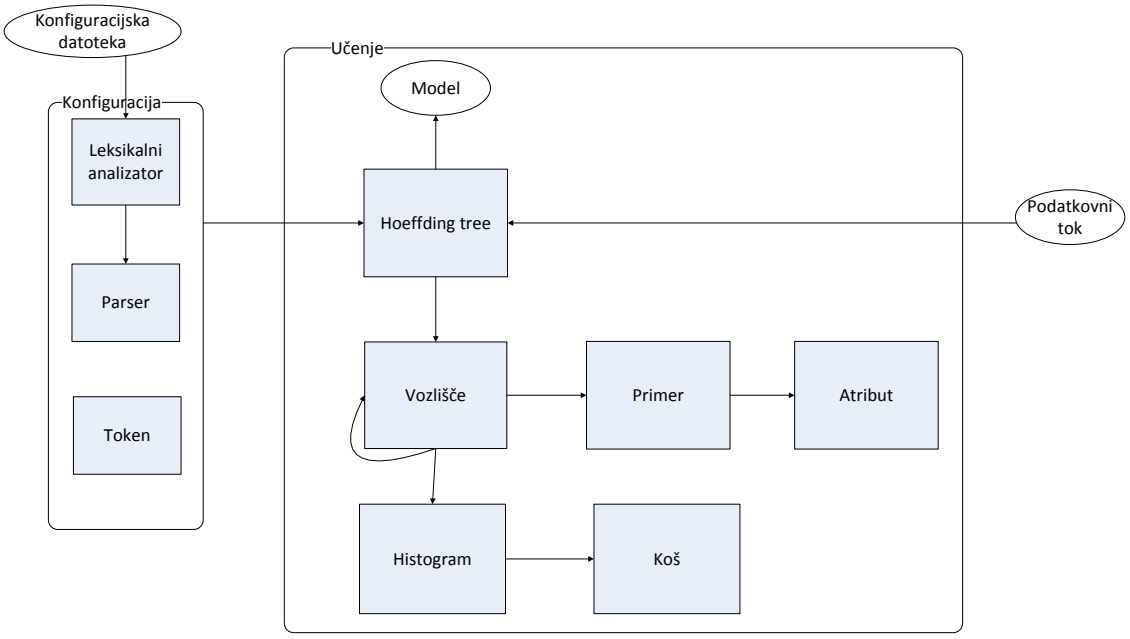
Na sliki 6.1 vidimo, da program pričakuje konfiguracijsko datoteko, v kateri opišemo podatkovni tok. Najprej navedemo seznam atributov v enakem vrstnem redu kot so le-ti v podatkovnem toku in nato za vsak atribut posebej navedemo tip, ki je ali numeričen ali diskreten — za diskretne atribute definiramo tudi zalogo vrednosti. Formalna LR gramatika [Aho et al., 2006] za konfiguracijsko datoteko je izredno preprosta in jo zato vključujemo:

$$
\begin{aligned}
id &\rightarrow (alpha \mid num \mid \_ \mid = \mid < \mid >)^{+} \\
lst &\rightarrow lst, id \mid id \\
hdr &\rightarrow \textbf{dataFormat} : (lst) \\
type &\rightarrow id : \textbf{discrete}(lst) \mid id : \textbf{numeric} \\
desc &\rightarrow desc\ type \mid type \\
S &\rightarrow hdr\ desc
\end{aligned}
$$

Pri tem je *alpha* množica malih in velikih črk in *num* množica $\{0, 1, 2, \ldots, 9\}$, začetni nekončni simbol pa je S. Na podlagi konfiguracijske datoteke program med inicializacijo zgradi interne podatkovne strukture, ki pohitrijo delovanje. Konkreten primer konfiguracijske datoteke najdemo v poglavju 7.

Parameter	Opis parametra
GracePeriod	Na vsake koliko učnih primerov naj se preračunajo ocene atributov.
SplitConfidence	Stopnja zaupanja, da je atribut z najvišjo vzorčno oceno res najboljši.
TieBreaking	Kdaj dva atributa smatramo za praktično enako dobra.
WindowSize	Število učnih primerov, ki jih hranimo v pomnilniku.
DriftCheck	Na vsake koliko učnih primerov preverjamo veljavnost razcepov.
ConfigNm	Pot do konfiguracijske datoteke.
AttrHeuristic	Hevristika za ocenjevanje atributov.
ConceptDriftP	Poženi varianto VFDT ali CVFDT algoritma.

Tabela 6.1: Glavni parametri programa.



Slika 6.1: Visokonivojska arhitektura implementacije. Pravokotniki predstavljajo osnovne razrede — imamo okoli 10 razredov — povezave pa medsebojne odvisnosti. Povezava od enega razreda do drugega pomeni, da prvi uporablja drugega. Razred vozlišče kaže sam nase, ker hrani vektor otrok, razred histogram pa kaže na razred koš, ker je histogram sestavljen iz košev. Izjema je token, ki si ga podajata leksikalni analizator in razčlenjevalnik (angl. parser); povezav zaradi preglednosti v tem primeru ne narišemo. Elipse predstavljajo vhodne in izhodne datoteke — konfiguracijska datoteka in podatkovni tok sta vhodni datoteki, model pa izhodna.

Ko program procesira konfiguracijsko datoteko in izvede inicializacijo, je pripravljen na učenje. Glavni parametri, ki jih program vzame iz ukazne vrstice in uporablja za učenje, so navedeni v tabeli 6.1.

Celoten "sistem" je "iz ničle" implementiran v C++ in uporablja le varianto standardne knjižnice, GLib[1]. Tako sta tudi leksikalni analizator in parser napisana ročno. Za več o uporabi glej poglavje 7.

6.2 Mere za ocenjevanje atributov

Naj bo X diskretna naključna spremenljivka s k vrednostmi. Potem rečemo, da je preslikava $\varphi : [0,1]^k \to \mathbb{R}$ *mera nečistoče*, če velja:

(i) φ je nenegativna,

(ii) φ zavzame minimalno vrednost natanko takrat, kadar je $\mathbb{P}(X = x_i) = 1$ za neko vrednost x_i,

(iii) φ zavzame maksimalno vrednost natanko takrat, kadar velja $\mathbb{P}(X = x_i) = 1/k$ za vse x_i,

(iv) φ je simetrična funkcija,

(v) φ je gladka funkcija.

Za dano učno množico E je razredna spremenljivka porazdeljena po

$$\begin{pmatrix} 1 & 2 & \cdots & c \\ p_1 & p_2 & \cdots & p_c \end{pmatrix},$$

pri čemer je p_i verjetnost i-tega razreda.

Oceno atributa A v splošnem definiramo kot pričakovano zmanjšanje nečistoče porazdelitve razredne spremenljivke po particioniranju učne množice E glede na vrednosti atributa A. Formalno, definiramo

$$\Delta\varphi(A, E) := \varphi(E) - \sum_{i=1}^{d} p_i \varphi(E_i), \tag{6.1}$$

pri čemer postavimo $E_i := \{x \in E : A(x) = a_i\}$.

V naslednjih podrazdelkih definiramo in opišemo dve meri nečistoče, informacijski dobitek in Gini indeks — edini meri, ki smo ju implementirali in razširili za uporabo na spremenljivih podatkovnih tokovih.

6.2.1 Računanje hevrističnih ocen.

Omejimo se na računanje informacijskega dobitka in Gini indeksa iz zadostne statistike.

Računanje informacijskega dobitka. Naj bo $C(A, V, R)$ tabela števcev, naj bo A atribut in naj bo a_i vrednost tega atributa. Potem je

$$|E_i| = \sum_{j=1}^{c} C(A, a_i, c_j),$$

pri čemer c_j označuje j-ti razred. Pri tem je $E_i := \{x \in E : A(x) = a_i\}$ za atribut A in njegovo i-to vrednost a_i. Informacijski dobitek je definiran kot pričakovano zmanjšanje entropije po particioniranju učne množice glede na vrednosti atributa A. Formalno, definirajmo

$$G(A) := H(E) - \sum_{i=1}^{d} p_i H(E_i), \tag{6.2}$$

[1]Avtor je Marko Grobelnik z ostalimi z odseka za umetno inteligenco na IJS.

pri čemer lahko izračunamo

$$H(E_i) = \sum_{k=1}^{c} r_k \log_2 r_k,$$

kjer je $r_k := C(A, a_i, c_k)/|E_i|$. V vsakem listu iščemo atribut z največjo hevristično oceno:

$$A^\star := \arg\max_A G(A).$$

Glej algoritem 8 za podrobnosti. Glej [Kononenko and Robnik-Šikonja, 2010] za dokaz, da je entropija res mera nečistoče.

Algoritem 8 Izračun informacijskega dobitka na podlagi zadostne statistike.

Naj bo $h := 0$ entropija učne množice
for $i := 1$ **to** c **do** `// Izračunaj` $H(E)$
    Naj bo $p := P[i]/n$
    $h := h - p \log_2 p$
for $j := 1$ **to** m **do** `// Za vsako od` m `vrednosti atributa` A
    Naj bo $e_j := 0$
    for $i = 1$ **to** c **do** `// Izračunaj` $|E_j|$
        $e_j := e_j + C(A, a_j, c_k)$
    Naj bo $p_j := e_j/n$
    Naj bo $h_j := 0$ entropija
    for $i := 1$ **to** c **do** `// Izračunaj` $H(E_j)$
        Naj bo $p_i := C(A, a_j, c_i)/e_j$
        $h_j := h_j - p_i \log_2 p_i$
    $h := h - p_j h_j$
return h `// Vrni` $\overline{G}(A)$

Računanje Gini indeksa. Intuitivno, Ginijeva mera nečistoče pove, kolikokrat bi naključno izbran učni primer napačno klasificirali, če bi ga klasificirali naključno glede na porazdelitev razredov v podmnožici. Za izračun Gini indeksa na podlagi zadostne statistike predlagamo algoritem 9. Glej [Kononenko and Robnik-Šikonja, 2010] za dokaz, da sta apriorni Gini indeks in pričakovani Gini indeks res meri nečistoče.

Apriorni Ginijev indeks (6.3) meri razliko med verjetnostnimi porazdelitvami vrednosti atributa. Definiramo ga kot

$$\mathrm{Gini}(E) := 1 - \sum_{k=1}^{c} p_k^2. \tag{6.3}$$

Ginijev dobitek definiramo kot pričakovano zmanjšanje razlike

$$\mathrm{GiniGain}(A, E) := \mathrm{Gini}(E) - \sum_{i=1}^{d} p_i \,\mathrm{Gini}(E_i). \tag{6.4}$$

6.3 Večvrednostni diskretni atributi in numerični atributi

V tem razdelku navedemo izbrane postopke za particioniranje večvrednostnih atributov in diskretizacijo numeričnih atributov na podatkovnih tokovih.

Algoritem 9 Izračun Gini dobitka na podlagi zadostne statistike.

Vhod: Algoritem vzame atribut A, zadostno statistiko lista C, in porazdelitev po razredih P.
Izhod: Vrne Gini dobitek GiniGain(A).

```
1: Naj bo g := 1 Gini indeks učne množice
2: for i := 1 to c do // Izračunaj Gini(E)
3:     Naj bo p := P[i]/n
4:     g := g − p^2
5: for j := 1 to m do // Za vsako od m vrednosti atributa A
6:     Naj bo e_j := 0
7:     for i = 1 to c do // Izračunaj |E_j|
8:         e_j := e_j + C(A, a_j, c_k)
9:     Naj bo p_j := e_j/n
10:    Naj bo g_j := 1 Gini indeks
11:    for i := 1 to c do // Izračunaj Gini(E_j)
12:        Naj bo p_i := C(A, a_j, c_i)/e_j
13:        g_j := g_j − p_i^2
14:    g := g − p_j g_j
   return g // Vrni GiniGain(A)
```

6.3.1 Particioniranje večvrednostnih diskretnih atributov

Naj bo A diskreten d-vrednostni atribut. Z združevanjem ga želimo spremeniti v q-vrednostnega za $q < d$. Število načinov za particioniranje d-množice v q nepraznih podmnožic štejejo Stirlingova števila druge vrste [Graham et al., 1994]. Naj bo $\left\{ {n \atop k} \right\}$ število particioniranj n-množice na k nepraznih podmnožic. Potem velja

$$\sum_{k=0}^{n} \left\{ {n \atop k} \right\} = B_n,$$

pri čemer B_n označuje n-to Bellovo število. Že za binarizacijo d-vrednostnega atributa imamo torej $\left\{ {d \atop 2} \right\} = 2^{d-1} - 1$ možnosti. Preiskovanje vseh možnih particioniranj je torej brezupno, zato v paketnem učenju uporabljamo hevristike (glej npr. [Coppersmith et al., 1999]). To pomeni, da iščemo nov q-vrednostni atribut $\widehat{A}$ z zalogo $D_1, D_2, \ldots, D_q$, da velja

$$D_1 \cup D_2 \cup \ldots \cup D_q = D \qquad\qquad D_i \cap D_j = \emptyset \text{ za } i \neq j,$$

pri čemer so D_i neprazne podmnožice vrednosti atributa A. V literaturi nismo zasledili postopkov za particioniranje večvrednostnih atributov na podatkovnih tokovih.

6.3.2 Diskretizacija numeričnih atributov

Glej [Pfahringer et al., 2008] za pregled pristopov k obravnavanju numeričnih atributov v inkrementalnem učenju odločitvenih dreves. Tukaj se osredotočimo na najbolj enostaven pristop: aproksimiraj porazdelitev razrednih oznak s histogramom z N koši, inicializiranimi z realnimi vrednostmi, in poglej, katero "razbitje" histograma maksimizira dano hevristiko.

Ocenjevanje porazdelitve vrednosti s histogramom

V vsakem vozlišču za vsak numeričen atribut hranimo histogram, ki ga inicializiramo z unikatnimi vrednostmi prvih $N := 100$ primerov. Vsak naslednji primer v danem histogramu posodobimo zadostno statistiko — število učnih primerov označenih s k-to razredno oznako, ki za atribut i zavzamejo j-to vrednost za vse kombinacije i, j, in k — košu, ki je inicializiran z vrednostjo, ki je najbližja vrednosti trenutnega učnega primera. Postopek je formalno prikazan v algoritmu 10.

Ta pristop ima vsaj tri resne pomanjkljivosti:

- odvisen je od vrstnega reda primerov v toku, kar pomeni, da porazdelitev lahko postane asimetrična (angl. skewed),
- parameter N — število košev — je uporabniško definiran in se razlikuje od scenarija do scenarija,
- podatkovni tokovi so inherentno spremenljivi, kar pomeni, da inicialne vrednosti histogramov kmalu niso več aktualne — lahko sestavimo patološki podatkovni tok, kjer se skoraj vse vrednosti akumulirajo v en sam koš histograma.

Ena od idej za nadaljnje delo je, da bi nad histogramom delali preprosto varianto razvrščanja kot [Brank, 2013]. Kadar sta dva koša praktično prazna, ju pozabimo in dodamo dva nova; kadar kadar sta si dva koša v nekem smislu preveč podobna, ju združimo v en sam koš in dodamo novega; prilagajamo tudi število košev.

Ta postopek ni primeren, kadar želimo pozabljati primere — histogram se lahko medtem, ko je primer v drsečem oknu, spremeni. Zato vsakemu košu dodelimo monotono naraščajoč ID, podobno kot to delamo za liste drevesa. Kadar primer akumuliramo v vsa vozlišča, mu dodelimo največjega izmed IDjev košev, v katere je bil akumuliran. Ta informacija je dovolj za pravilno pozabljanje.

Algoritem 10 Uporaba histograma v enem od vozlišč za dan numerični atribut.

Vhod: Algoritem vzame podatkovni tok S in število košev N.
Izhod: Posodablja histogram kot ga uporabljamo za binarizacijo numeričnih atributov.

```
for x ∈ S do// Za vsak primer podatkovnega toka
    Naj bo v := A(x) vrednost numeričnega atributa A
    if histogram ima N košev ali obstaja koš, inicializiran z vrednostjo v then
        Naj bo i* := arg min_{i∈[N]} (v − v_i)^2 indeks najbližjega koša
        Posodobi zadostno statistiko koša B_{i*}
    else
        Urejeno dodaj histogramu prazen koš B, inicializiran z vrednostjo v
```

Mislimo si, da histogram presekamo na dva dela na enem izmed košev in naredimo unijo košev na levi in na desni. Informacijski dobitek takega reza je razlika entropije celega histograma in uteženе vsote entropij leve in desne unije košev histograma. Iščemo tisti rez, ki maksimizira informacijski dobitek. Postopek je formalno opisan v algoritmu 11, intuitivno pa je prikazan na sliki 6.2.

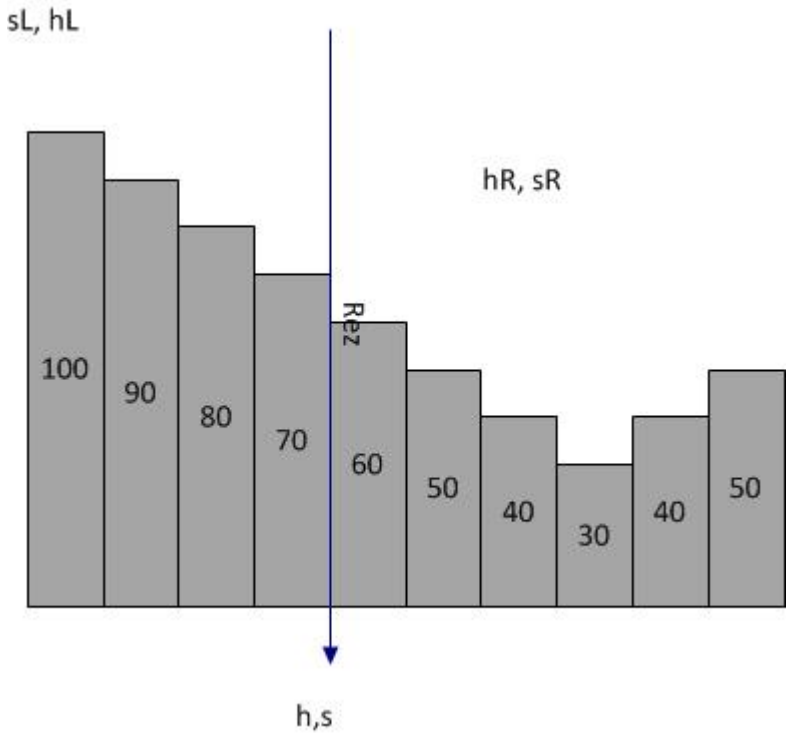
Drugi pristopi

Gama in ostali [Gama et al., 2003] predlagajo postopek za obravnavo zveznih atributov; glej tudi [Pfahringer et al., 2008]. Ta pristop omenjamo, ker je osnova za diskretizacijo v FMIT-DD [Ikonomovska et al., 2011].

V vsakem listu drevesa za vsak diskretni atribut hranimo vektor porazdelitve razrednih oznak učnih primerov, ki so padli v ta list, in za vsak zvezni atribut hranimo binarno drevo, ki ima v vsakem vozlišču pragovno vrednost i — to je vrednost enega od učnih primerov, ki je padel v ta list — in dva k-vektorja L in H, ki štejeta število primerov manjših ali enakih i in strogo večjih od i, ki so šli skozi to vozlišče in so imeli določeno razredno oznako. Ko nov učni primer doseže list, posodobimo vsa binarna drevesa — po eno za vsak zvezni atribut.

Informacijski dobitek zveznega atributa izračunamo za vse možne binarizacije — pragovno vrednost izberemo iz množice vrednosti, ki jih je atribut zavzel do sedaj. Formalno definiramo informacijski dobitek kot

$$\mathrm{info}(A_j(i)) := \mathbb{P}(A_j \leq i)\,\mathrm{iLo}(A_j(i)) + \mathbb{P}(A_j > i)\,\mathrm{iHi}(A_j(i)),$$



Slika 6.2: Velika slika postopka binarizacije numeričnih atributov. Na sliki velja $s_L := 340$ in $s_H := 270$, kar pomeni, da je informacijski dobitek tega reza enak $h - \frac{340}{340+270} h_L - \frac{270}{340+270} h_R$. To ponovimo za vse koše in binariziramo atribut v točki, ki maksimizira informacijski dobitek — ta točka je vedno vrednost, s katero je inicializiran eden od košev histograma.

Algoritem 11 Binarizacija numeričnih atributov pri klasifikaciji.

Vhod: Algoritem vzame histogram z n koši.
Izhod: Vrne indeks koša, ki maksimizira informacijski dobitek.

Naj bodo $B_1, B_2, \ldots, B_n$ koši histograma in naj bodo $v_1, v_2, \ldots, v_n$ pozitivna realna števila, s katerimi so koši inicializirani
Naj bodo $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ vsebine košev in naj bodo $s_1, s_2, \ldots, s_n$ vsote elementov košev
Naj bosta h_L in s_L entropija in vsota elementov "unije" levega dela histograma in naj bosta h_H in s_H entropija in vsota elementov "unije" desnega dela histograma
for $i \in [n]$ **do** `// Izračunaj entropijo unije vseh košev histograma`
    $\mathbf{x}_L := \mathbf{x}_L + \mathbf{x}_i$
    $s_L := s_L + s_i$
Inicializiraj $h_H := s_H := 0$ in $s := s_1 + s_2 + \ldots + s_n$
Očitno je vsebina unije vseh košev histograma $\mathbf{x} := \mathbf{x}_L$
for $i := n$ **to** 1 **do** `// Poišči točko, ki maksimizira informacijski dobitek`
    Posodobi $\mathbf{x}_L := \mathbf{x}_L - \mathbf{x}_i$ in $\mathbf{x}_H := \mathbf{x}_H + \mathbf{x}_i$
    Zapomni si dobitek v tej točki $g_i := \textsc{InfoGain}(\mathbf{x}, \mathbf{x}_L, \mathbf{x}_H)$
return $\arg\max_{i \in [n]} g_i$ `// Vrni indeks koša, ki maksimizira informacijski dobitek`
function $\textsc{Entropy}(\mathbf{x})$
    Naj bo $s := 0$ vsota komponent $\mathbf{x} \in \mathbb{R}^C$ za naraven $C > 0$
    for $i := 0$ **to** C **do**
        $s := s + \mathbf{x}[i]$
    Naj bo $h := 0$ entropija
    for $i := 0$ **to** C **do**
        $h := h - \frac{\mathbf{x}[i]}{s} \log_2 \frac{\mathbf{x}[i]}{s}$
    return h
function $\textsc{InfoGain}(\mathbf{x}, \mathbf{x}_L, \mathbf{x}_H)$
    Naj bodo $s := s_L := s_H := 0$ vsote komponent zgornjih vektorjev iz $\mathbb{R}^C$ za naraven $C > 0$
    for $i := 0$ **to** C **do**
        $s := s + \mathbf{x}[i]$
        $s_L := s_L + \mathbf{x}_L[i]$
        $s_H := s_H + \mathbf{x}_H[i]$
    Naj bodo $h := \textsc{Entropy}(\mathbf{x})$, $h_L := \textsc{Entropy}(\mathbf{x}_L)$ in $h_H := \textsc{Entropy}(\mathbf{x}_H)$ entropije
    return $h - \frac{s_L}{s} h_L - \frac{s_H}{s} h_H$

pri čemer postavimo

$$\mathrm{iLo}(A_j(i)) := -\sum_k \mathbb{P}(K = k | A_j \leq i) \log_2 \mathbb{P}(K = k | A_j \leq i),$$
$$\mathrm{iHi}(A_j(i)) := -\sum_k \mathbb{P}(K = k | A_j > i) \log_2 \mathbb{P}(K = k | A_j > i).$$

Ta pristop ni primeren za podatkovne tokove, ker porabi potencialno neomejeno pomnilnika.

6.4 Klasifikacija v listih

V času tega pisanja v listih implementirano večinski klasifikator in Navinega Bayesa.

Večinski klasifikator. V tem primeru iščemo razredno oznako, ki se med primeri iz množice E največkrat pojavi, tj.

$$\mathsf{Maj}(E) := \arg\max_{i \in [c]} \ \mathbb{P}(c_i),$$

pri čemer verjetnosti razredov ocenimo iz podatkov z relativnimi frekvencami.

Naivni Bayesov klasifikator. Tukaj iščemo razredno oznako, ki je za dan primer najbolj verjetna, tj.

$$\begin{aligned} y^\star &:= \arg\max_{y_k} \ \frac{\mathbb{P}(Y = y_k) \prod_{i=1}^{n} \mathbb{P}\left(X_i | Y = y_k\right)}{\sum_{j=1}^{c} \left(\mathbb{P}\left(Y = y_j\right) \prod_{i=1}^{n} \mathbb{P}\left(X_i | Y = y_j\right) \right)} \\ &= \arg\max_{y_k} \ \mathbb{P}(Y = y_k) \prod_{i=1}^{n} \mathbb{P}\left(X_i | Y = y_k\right), \end{aligned} \tag{6.5}$$

pri čemer verjetnosti ocenimo iz podatkov. (Naivni Bayes predpostavlja, da so atribut med seboj neodvisni.) Za ocenjevanje pogojnih verjetnosti uporabljamo m-oceno, za ocenjevanje verjetnosti razreda pa Laplacevo oceno. Glej [Cestnik, 1991] za podrobnosti.

6.5 Izvoz in vizualizacija

V času tega pisanja implementiramo izvoz trenutnega modela v XML, DOT in JSON. Algoritem 12 skicira izvoz odločitvenega drevesa v XML formatu. (Dejanska implementacija ni rekurzivna.) Glej

Algoritem 12 Izvoz odločitvenega drevesa v XML format.

Vhod: Algoritem vzame odločitveno drevo T.
Izhod: Izpiše drevo v XML formatu.

1: Naj bo `CurrNode` := `T.Root` koren drevesa `T`
2: **if** `CurrNode` je list **then** `<leaf></leaf>` **return** `// Končaj`
3: **for** `CurrNode` ∈ `T.Children` **do** `// Za vsakega sina <node>`
4: Poženi algoritem 12 na poddrevesu s korenom `CurrNode` `// Rekurzivni klic </node>`

poglavje 7 za konkreten primer.

7 Preprost primer uporabe

V tem poglavju ilustriramo uporabo naše implementacije[1] na preprostem scenariju.

7.1 Primer na nespremenljivih in spremenljivih Titanic podatkih

V tem razdelku uporabimo našo implementacijo na TITANIC-2.2M podatkih. Implementacija sloni na GLib[2] knjižnici, ki v grobem ponuja podobno funkcionalnost kot standardna C++ knjižnica in implementira precej dodatnih algoritmov in podatkovnih struktur, ki pa jih skoraj ne uporabljamo.

Koda 7.1 je primer preproste konfiguracijske datoteke. Prva vrstica algoritmu pove zaporedje vrednosti atributov v podatkovnem toku; vsaka naslednja vrstica definira atribut, ki je lahko diskreten (ključna beseda `discrete`) ali numeričen (ključna beseda `numeric`); za diskretne atribute naštejemo tudi zalogo vrednosti.

Koda 7.1: Primer konfiguracijske datoteke.

```
1 # Primer konfiguracijske datoteke za Titanic dataset
2 dataFormat: (status,age,sex,survived)
3 status: discrete(first,second,third,crew)
4 age: discrete(adult,child)
5 sex: discrete(male,female)
6 survived: discrete(yes,no)
```

Koda 7.2 je primer uporabe implementacije v C++. Skozi `TEnv` navedemo pot do konfiguracijske datoteke iz kode 7.1 in definiramo privzete parametre učenca;potem ustvarimo nov objekt `ht` s katerim se preprosto inkrementalno učimo odločitveno drevo. Koda ilustrira procesiranje učnih primerov, uporabo modela za napovedovanje, in izvoz modela v XML, JSON in DOT formate.

[1] Izvorna koda je dostopna na `https://github.com/blazs/HoeffdingTree`.
[2] Avtor knjižnice je Marko Grobelnik z ostalimi z Laboratorija za umetno inteligenco na IJS.

Koda 7.2: Primer uporabe implementacije.

```
#include "hoeffding.h"

using namespace TDatastream;

void ProcessData(const TStr& FileNm, PHoeffdingTree HoeffdingTree);

int main(int argc, char** argv) {
  try {
    Env = TEnv(argc, argv, TNotify::StdNotify);
    const bool ConceptDriftP = Env.IsArgStr("drift");
    const double SplitConfidence = \
      Env.GetIfArgPrefixFlt("-splitConfidence:", 1e-6, "Split confidence");
    const double TieBreaking = \
      Env.GetIfArgPrefixFlt("-tieBreaking:", 0.05, "Tie breaking");
    const TStr ConfigFNm = \
      Env.GetIfArgPrefixStr("-config:", "titanic.config", "Config file");
    const TStr AttrHeuristic = \
      Env.GetIfArgPrefixStr("-attrEval:", "InfoGain", "Attribute evaluation");
    const int GracePeriod = \
      Env.GetIfArgPrefixInt("-gracePeriod:", 300, "Grace period");
    const int DriftCheck = \
      Env.GetIfArgPrefixInt("-driftCheck:", 20000, "Drift check");
    const int WindowSize = \
      Env.GetIfArgPrefixInt("-windowSize:", 100000, "Window size");

    PHoeffdingTree ht = THoeffdingTree::New("docs/" + ConfigFNm, GracePeriod,
        SplitConfidence, TieBreaking, DriftCheck, WindowSize);
    TTmProfiler Prof;
    Prof.AddTimer("HoeffdingTree");
    Prof.StartTimer(0);
    ProcessData("data/titanic-220M.dat", ht);
    Prof.StopTimer(0);
    printf("Cas ucenja = %f sekund\n", Prof.GetTimerSec(0));

    int Label = ht->Classify("first,adult,female,no");
    ht->Export("exports/titanic-220M.xml"); // defaults to TExportType::XML
    ht->Export("exports/titanic-220M.gv", TExportType::DOT);
    ht->Export("exports/titanic-220M.json", TExportType::JSON);
  } catch(PExcept Except) {
    printf("%s\n", TStr("[Error] "+Except->GetMsgStr()).CStr());
    return 2;
  }
  return 0;
}

// process data line-by-line
void ProcessData(const TStr& FileNm, PHoeffdingTree HoeffdingTree) {
  Assert(TFile::Exists(FileNm));
  TFIn FIn(FileNm); TStr Line;
  while(FIn.GetNextLn(Line)) { HoeffdingTree->Process(Line, ','); }
}
```

Koda 7.3 prikazuje uporabo našega algoritma skozi QMiner[3] Javascript API. Izpostavimo, da ta funkcionalnost še ni v celoti implementirana — integracija algoritma v QMiner [Fortuna and Rupnik, 2014] in implementacija pripadajočega Javascript vmesnika sta plana za bližnjo prihodnost.

[3]Avtor QMiner platforme je Blaž Fortuna z Laboratorija za umetno inteligenco na IJS.

Koda 7.3: Primer uporabe skozi QMiner Javascript API.

```
// Example using HoeffdingTree via QMiner Javascript API
console.say("HoeffdingTree test");

var analytics = require('analytics');
var assert = require('assert.js');

// describe data stream
var titanicConfig = {
  "dataFormat": ["status", "age", "sex", "survived"],
  "status": { "type": "discrete", "values": ["first", "second", "third", "crew"] },
  "age": { "type": "discrete", "values": ["child", "adult"] },
  "sex": { "type": "discrete", "values": ["male", "female"] },
  "survived": { "type": "discrete", "values": ["yes", "no"] }
};

// algorithm parameters
var htParams = {
  "gracePeriod": 300,
  "splitConfidence": 1e-6,
  "tieBreaking": 0.01,
  "driftCheck": 1000,
  "windowSize": 100000,
  "conceptDriftP": false
};

// create a new learner
var ht = analytics.newHoeffdingTree(titanicConfig, htParams);

// train the model
var streamData = fs.openRead("./sandbox/hoeffdingtree/titanic-220K.dat");
while(!streamData.eof) {
  var line = streamData.getNextLn().split(",");
  // get discrete attributes
  var example_discrete = line.slice(0,3);
  // get numeric attributes
  var example_numeric = [];
  // get target
  var target = line[3];
  // update the model
  ht.process(example_discrete, example_numeric, target);
}

// use the model
var label = ht.classify(["first", "adult", "female"], []);
console.say("Were high society women likely to survive? " + label);

// export the model
ht.exportModel({ "file": "./sandbox/hoeffdingtree/titanic.gv", "type": "DOT" });
```

Slika 7.1 prikazuje sliko modela generirano z DOT[4] na podlagi izvoženega modela v DOT formatu

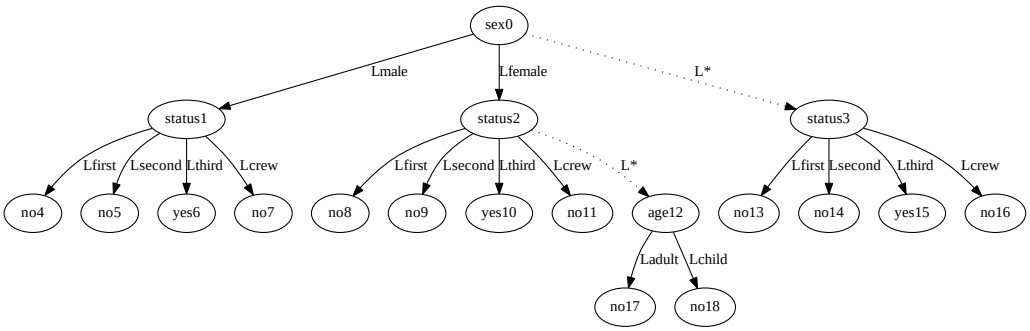
[4]DOT je program, ki na podlagi preprostega opisnega jezika DOT generira slike grafov.

iz kode 7.2.

Koda 7.4: Primer izvoženega modela v DOT.

```
digraph dt_fig {
  sex0 -> status1 [label="Lmale"];
  sex0 -> status2 [label="Lfemale"];
  sex0 -> status3 [label="L*",style="dotted"];
  status1 -> "no4" [label="Lfirst"];
  status1 -> "no5" [label="Lsecond"];
  status1 -> "yes6" [label="Lthird"];
  status1 -> "no7" [label="Lcrew"];
  status2 -> "no8" [label="Lfirst"];
  status2 -> "no9" [label="Lsecond"];
  status2 -> "yes10" [label="Lthird"];
  status2 -> "no11" [label="Lcrew"];
  status2 -> age12 [label="L*",style="dotted"];
  status3 -> "no13" [label="Lfirst"];
  status3 -> "no14" [label="Lsecond"];
  status3 -> "yes15" [label="Lthird"];
  status3 -> "no16" [label="Lcrew"];
  age12 -> "no17" [label="Ladult"];
  age12 -> "no18" [label="Lchild"];
}
```

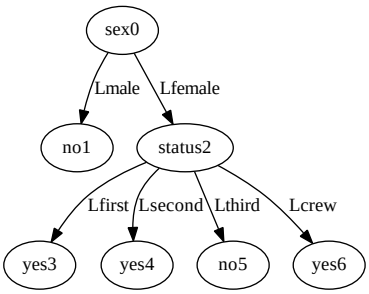
Slika 7.1: Slika izvoženega drevesa, generirana iz kode 7.4.



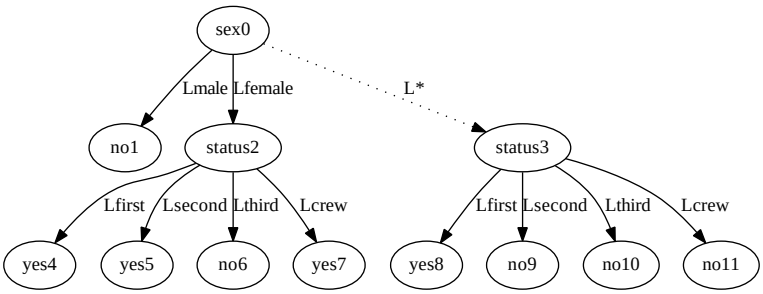
Podatkom Titanic-2M smo na konec dodali primere, v katerih se utopijo vsi potniki, razen tistih iz tretjega razreda. Slika 7.2 prikazuje evolucijo modela med učenjem z algoritmom 2 na opisanih podatkih. Slike smo izvozili, kadar je učenec začel gojiti alternativno drevo ali pa zamenjal glavno poddrevo z enim od alternativnih dreves v kateremkoli izmed vozlišč. Prvi štirje modeli so klasični za Titanic-2M, naslednja dva modela odražata okno s pomešanimi novimi in starimi primeri, model s slike 7.2g je iz trenutka, ko učenec opazi, da je atribut status obetavnejši od starosti, slika 7.2h pa prikazuje končni model, kjer algoritem zamenja trenutno poddrevo z alternativnim drevesom z atributom status v korenu.

7.2 Komentar

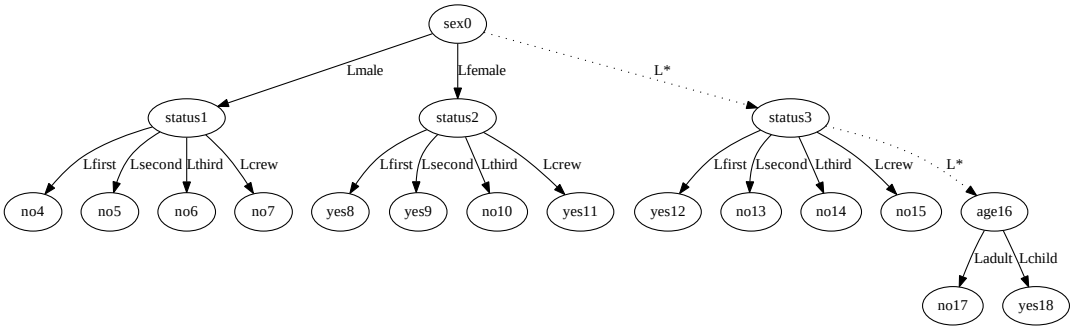
Ena od idej za nadaljnje delo je razširitev jezika za konfiguracijo z ukazi za določanje ranga numeričnih atributov — v praksi ponavadi vemo, da bo numeričen atribut za, na primer, starost osebe pozitiven in manjši od, recimo, 200. Druga ideja je razširitev implementacije s funkcionalnostjo za dinamično dodajanja vrednosti diskretnih atributov — to na primer pomeni, da nam za atribut, ki predstavlja



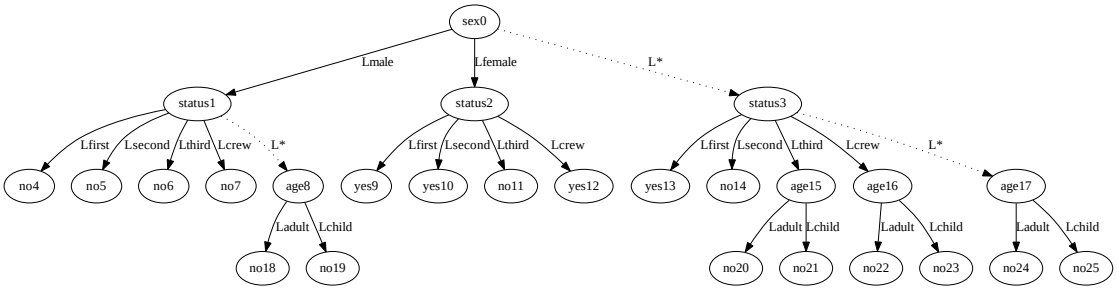
(a) Model 1.



(b) Model 2.

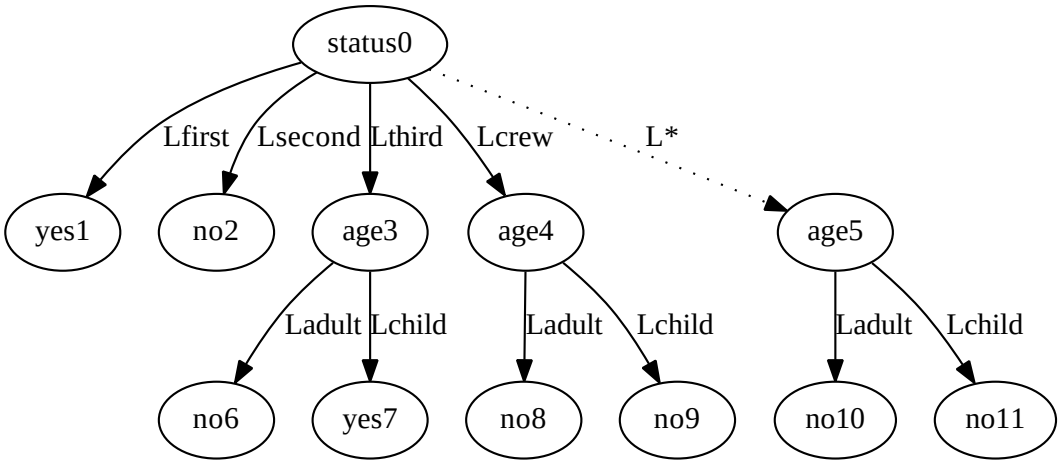


(c) Model 3.

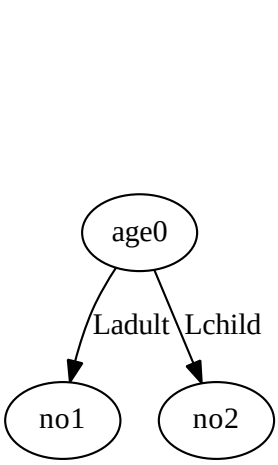


(d) Model 4.

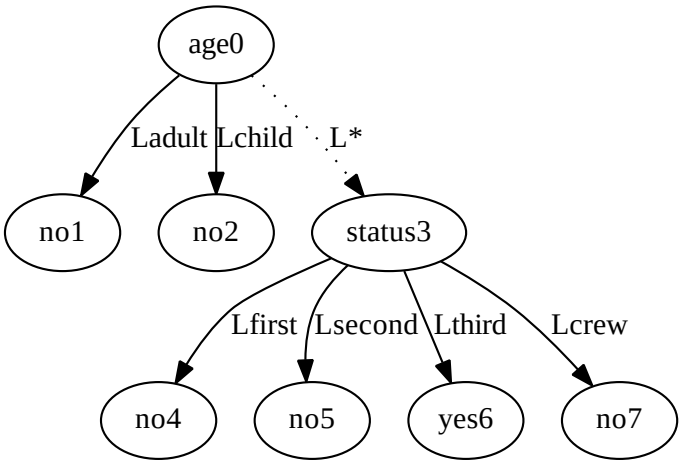
Slika 7.2: Evolucija modela med učenjem na spremenljivi varianti TITANIC-2M dataseta.



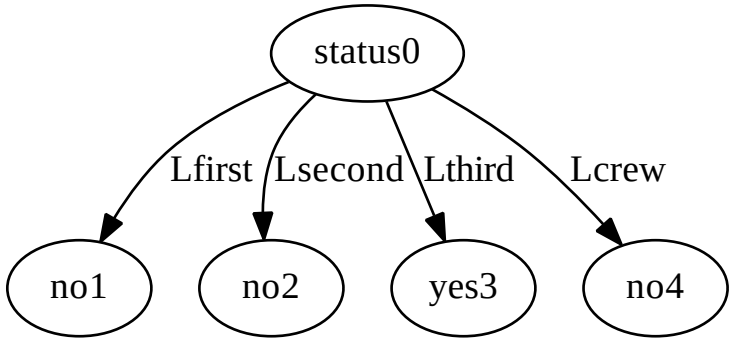
(e) Model 5.



(f) Model 6.



(g) Model 7.



(h) Model 8.

Slika 7.2: Evolucija modela med učenjem na spremenljivi varianti TITANIC-2M dataseta (nadaljevanje).

državo, ni potrebno našteti vseh držav v konfiguracijski datoteki.

Algoritem za diskretizacijo je sicer enostaven in efektiven, vendar po našem mnenju ni primeren za učenje na spremenljivih podatkovnih tokovih. Ideja je, da bi na histogramu delali varianto inkrementalnega k-means razvrščanja z združevanjem in cepljenjem košev, podobno kot [Brank, 2013].

8 Ocenjevanje uspešnosti učenja in primerjava algoritmov

V tem poglavju se ukvarjamo s problemi ocenjevanja uspešnosti učenja (razdelek 8.1) in primerjave učnih algoritmov (razdelek 8.2) na podatkovnih tokovih.

Za ocenjevanje uspešnosti učenja klasične mere niso uporabne, ker enakomerno utežijo napake preko celega podatkovnega toka. Zato ponavadi uporabljamo mere s pozabljanjem, da se za vsako časovno točko toka pridobi vrednost metrike, ki govori o uspešnosti toka v tistem danem trenutku, pri čemer je uspešnost bolj utežena s sedanjo vrednostjo in manj s prejšnjimi vrednostmi napak. Druga možnost je pristop z izločanjem učnih primerov (angl. holdout evaluation), kjer periodično — tipično na vsakih $10\,000 \leq k \leq 100\,000$ primerov — žrtvujemo $m := 2000$ primerov za evalvacijo.

Za primerjavo dveh algoritmov uporabljamo Q-statistiko, hitro in inkrementalno statistiko, definirano kot logaritem kvocienta izgub učencev v dani točki, ki "zvezno" primerja napovedno točnost teh dveh učnencev.

Pristopi, predstavljeni v nadaljevanju, so povzeti po [Gama et al., 2009, 2013]. Glej tudi [Bosnić et al., 2011] in [Rodrigues et al., 2012]. Ker gre za novo področje, nimamo ničesar podobnega [Demšar, 2006].

8.1 Ocenjevanje uspešnosti učenja

V sledečih podrazdelkih opišemo pristop z izločanjem učnih primerov (podrazdelek 8.1.1) in pristop z bledečimi faktorji (podrazdelek 8.1.2).

8.1.1 Ocenjevanje z izločanjem učnih primerov

Pri tem načinu periodično, ponavadi na vsake $10\,000 \leq k \leq 100\,000$ primerov, žrtvujemo $m := 2000$ primerov, ki jih uporabimo za evalvacijo trenutnega modela. Za dovolj velik m je taka cenilka nepristrana (angl. *unbiased*). Za m učnih primerov jo definiramo kot

$$H_m := \frac{1}{m} \sum_{i=1}^{m} L(y_i, \hat{y}_i).$$

To je najbolj preprost možen pristop, saj v paketni terminologiji žrtvovani primeri ustrezajo testni, ostali primeri pa učni množici.

8.1.2 Ocenjevanje uspešnosti učenja z bledečimi faktorji

Sedaj opišemo pristope, ki nimajo paketnih analogov — osredotočimo se na ocenjevanje uspešnosti z drsečimi okni in z bledečimi faktorji.

Drseča okna

Ta pristop je v času tega pisanja med najbolj popularnimi. Za drseče okno velikosti w v točki k definiramo napako kot

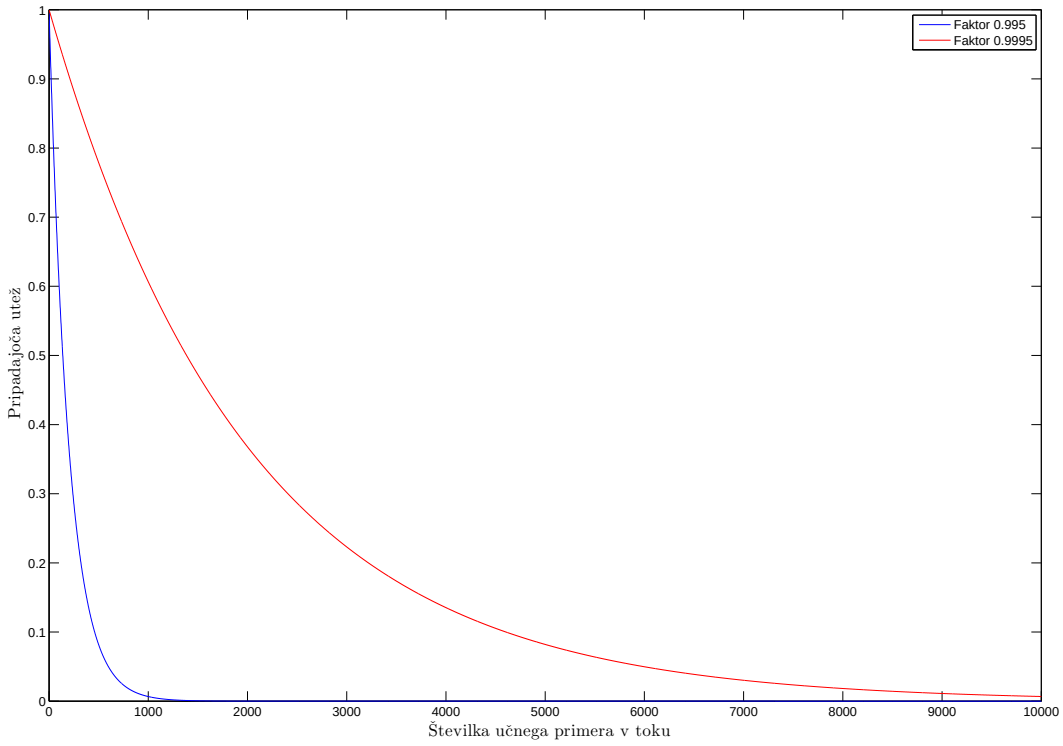
$$P_w(i) := \frac{1}{w} \sum_{i=k-w+1}^{k} L(y_i, \hat{y}_i).$$

Tako upoštevamo samo napake iz bližnje preteklosti; pri tem je prostorska zahtevnost $O(w)$. Intuicija je, da starejše napako niso relevantne, ker se je model medtem spremenil. Zdi se, da ima ta pristop vsaj dve pomanjkljivosti:

- parameter w za velikost okna je uporabniško definiran in se verjetno razlikuje od scenarija do scenarija, glede na stopnjo spreminjanja v podatkih,
- velikost okna ostane fiksna ne glede na spremembe v podatkih.

Bledeči faktorji

Tukaj je glavna ideja, da napake utežimo glede na njihovo starost. Slika 8.1 prikazuje uteži za $n := 10\,000$ primerov za bledeča faktorja $\alpha := 0.995$ in $\alpha := 0.9995$. Relevantnost starejših napak torej definiramo prek bledečih faktorjev. Slednji so multiplikativni in ustrezajo eksponentnemu pozabljanju (slika 8.1).



Slika 8.1: Uteži za 10 000 primerov za bledeča faktorja $\alpha := 0.995$ in $\alpha := 0.9995$.

Naj bo $\alpha \in (0, 1]$ veliko večji od 0, recimo $\alpha \geq 0.99$. Za podatkovni tok x definirajmo bledečo vsoto v točki i kot

$$S_{x,\alpha}(i) := L(y_i, \hat{y}_i) + \alpha S_{x,\alpha}(i-1),$$

pri čemer postavimo $S_\alpha(1) := L(y_1, \hat{y}_1)$. Nadalje naj bo bledeči inkrement

$$N_\alpha(i) := 1 + \alpha N_\alpha(i-1),$$

pri čemer postavimo $N_\alpha(1) := 1$.

Tako je ocenjena pričakovana napaka (angl. *prequential error*), znana tudi kot "testiraj-potem-nauči" (angl. *test-then-train*), definirana kot

$$\begin{aligned} P_\alpha(i) &:= S_\alpha^A(n)/N^\alpha(n) \qquad (8.1)\\ &= \frac{\sum_{k=1}^{i} \alpha^{i-k} L(y_k, \hat{y}_k)}{\sum_{k=1}^{i} \alpha^{i-k}} \\ &= \left(\frac{1-\alpha^{i+1}}{1-\alpha} - 1\right)^{-1} \sum_{k=1}^{i} \alpha^{i-k-1} L(y_k, \hat{y}_k) \\ &= \frac{1-\alpha}{\alpha(1-\alpha^i)} \sum_{k=1}^{i} \alpha^{i-k} L(y_k, \hat{y}_k) \\ &= \frac{1-\alpha}{1-\alpha^i} \sum_{k=1}^{i} \alpha^{i-k-1} L(y_k, \hat{y}_k). \qquad (8.2) \end{aligned}$$

Vidimo, da je $P_\alpha(i)$ definiran podobno kot klasifikacijska točnost: gre za kvocient dejanske napake $S_\alpha^A(i)$ in največje možne napake[1] $N_\alpha(i)$, ki je preprosto delna vsota geometrijske vrste. (Taka oblika je bolj intuitivna od rekurzivnih formul, ki jih uporabljamo v praksi.)

Za razliko od drsečih oken imamo tukaj majhno konstantno prostorsko zahtevnost. Tudi tukaj imamo uporabniško definiran parameter α s katerim definiramo, kako stare napake so za nas še relevantne (slika 8.1).

Algoritem 13 Računanje napake z bledečimi faktorji.

Vhod: Vzame bledeči faktor $\alpha \in (0, 1]$ in napako $L(y_i, \hat{y}_i)$ pri i-tem primeru.
Izhod: Vrne cenilko za $P_\alpha(i)$.

```
1: function P(α, n)
2:     if n = 0 then
3:         Inicializiraj S_α(0) := N_α(0) := 0
4:         return 0
5:     else
6:         S_α(n+1) := L(y_n, ŷ_n) + αS_α(n)
7:         N_α(n+1) := 1 + αN_α(n)
8:         P_α(n+1) := S_α(n)/N_α(n)
       return P_α(n+1)
```

8.2 Primerjava dveh algoritmov na podatkovnem toku

V tem razdelku se ukvarjamo s primerjavo uspešnosti učenja dveh algoritmov na podatkovnem toku. Razlikovati želimo med naključnimi in nenaključnimi razlikami v meritvah eksperimentov.

Naj bosta S_α^A in S_α^B zaporedji napak učencev A in B v vseh točkah podatkovnega toka, definirani kot $S_A^\alpha(i) := L_A(y_i, \hat{y}_i) + \alpha S_A^\alpha(i-1)$ za $i > 0$ in $S_0^A := 1$, za učenca A. Potem za $\alpha \in (0, 1]$ in $i \in \mathbb{N}$ definiramo Q-statistiko kot preslikavo $Q : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}$, dano s predpisom[2]

$$Q(S_A^\alpha(i), S_B^\alpha(i)) := \log\left(S_A^\alpha(i), S_B^\alpha(i)\right), \qquad (8.3)$$

[1] Ta interpretacija velja za 0-1 izgubo.

[2] Za voljo preprostosti v nadaljevanju pišemo $Q_i^\alpha(A, B)$ namesto $Q(S_A^\alpha(i), S_B^\alpha(i))$.

ki jo lahko uporabimo s skoraj vsemi funkcijami izgub. Signal Q_i^α odraža relativno uspešnost obeh modelov v i-ti točki, njegovo vrednost pa interpretiramo kot moč razlike. Funkcija Q_i je tudi simetrična do predznaka natančno, ker je $\log(x/y) = -\log(y/x)$.

Vrednost Q-statistike interpretiramo tako, da pogledamo predznak in absolutno vrednost $Q_i^\alpha(A, B)$:

- $Q_i^\alpha(A, B) < 0$ pomeni, da je A boljši od B,
- $Q_i^\alpha(A, B) = 0$ pomeni, da ni razlike,
- $Q_i^\alpha(A, B) > 0$ pomeni, da je B boljši od A.

Absolutna vrednost $|Q_i^\alpha(A, B)|$ pove, koliko boljši je eden učenec od drugega — interpretiramo jo lahko kot moč razlike med učencema.

Naj bosta A in B učenca na danem podatkovnem toku in naj bo $\mathbf{q}_i := Q_i^\alpha(A, B)$ za $\mathbf{q} \in \mathbb{R}^n$ za $1 \leq i \leq n$. Naj bo $\alpha \in (0, 1)$ značilnost Wilcoxonovega testa [Wilcoxon, 1945]. To je paren neparametričen test, ki testira ničelno hipotezo, da je vektor $\mathbf{q}$ iz porazdelitve z ničelno mediano. Pri vseh poskusih privzamemo $\alpha := 0.01$. Ničelno hipotezo zavrnemo, če je p-vrednost — v grobem verjetnost ničelne hipoteze — strogo manjša od značilnosti testa α. Manjša kot je p-vrednost, manj verjetno je, da je ničelna hipoteza resnična.

V naslednjem poglavju primerjamo več parov algoritmov na istem podatkovnem toku, kar pomeni, da moramo — seveda to ni edina rešitev — uporabiti Bonferronijev popravek [Salzberg, 1997]. Naj bo α izbrana stopnja zaupanja in brez škode recimo, da primerjamo n parov algoritmov. Potem imamo $n\alpha$ možnosti za vsaj en značilen rezultat. Naj bo $\alpha^\star$ stopnja zaupanja za en eksperiment; potem je $1 - \alpha^\star$ verjetnost, da v tem eksperimentu nismo naredili napake (tipa I ali II). Če naredimo n neodvisnih eksperimentov pri tej stopnji zaupanja, potem je verjetnost, da pri nobenem nismo naredili napake enaka $(1 - \alpha^\star)^n$. Torej je verjetnost za vsaj eno napako enaka $1 - (1 - \alpha^\star)^n$, kar pomeni, da moramo za želeno stopnjo zaupanja $\alpha := 0.01$ izbrati $\alpha^\star$, ki reši neenačbo

$$1 - (1 - \alpha^\star)^n \leq \alpha.$$

Izkaže se, da je dovolj postaviti $\alpha^\star := \alpha/n$.[3]

[3]To je Bonferronijev popravek, ki sledi iz Bonferronijevih neenakosti, ki jih je dokazal Firenski matematik Carlo Emilio Bonferroni (1892–1933).

9 Rezultati

V tem poglavju navedemo rezultate naših implementacij na izbranih podatkovnih tokovih. Za vrednotenje uporabljamo uporabili metode iz [Gama et al., 2009, 2013] in [Bosnić et al., 2011] ter [Demšar, 2006] in [Salzberg, 1997].

Vse evalvacije so bile narejene na Microsoft Windows 7 platformi na Thinkpad 530T z 8GB delovnega pomnilnika in 4 jedrnim Intel i7 procesorjem.

Spodaj med sabo primerjamo več algoritmov, zato konzervativno postavimo $\alpha := 0.0001$, kar bi zadostovalo za 100 primerjav. Po Bonferroniju bi bilo dovolj $\alpha = 0.01/6$.

9.1 Napaka pri testiranju z izločanjem testnih primerov

V tem razdelku primerjamo učna algoritma VFDT in CVFDT, pri čemer v listih uporabljamo večinski klasifikator in Naivnega Bayesa.

Slika 9.1 prikazuje primerjavo algoritmov 1 (VFDT-MAJ) in 2 (CVFDT-MAJ) z večinskim klasifikatorjem na podatkih NYEL-NUM, pri čemer napovedujemo porabo za naslednjih pet minut. Na vsakih 10 000 primerov smo žrtvovali 2 000 primerov za izračun klasifikacijske točnosti. Izkaže se, da je algoritem 1 boljši ($p < 0.0001$).

Na sliki 9.2 je analogen scenarij za algoritma 1 (VFDT-NB) in 2 (CVFDT-NB) z Naivnim Bayesom v listih. V tem primeru je ne moremo povedati, kateri algoritem je boljši (velja $p = 0.6538$).

Na sliki 9.3 je analogen scenarij za algoritma CVFDT-NB in CVFDT-MAJ — eden ima v listih Naivnega Bayesa, drugi pa večinski klasifikator. Boljši je prvi algoritem CVFDT-NB (velja $p < 0.0001$).

9.2 Faktorji pozabljanja

V tem razdelku ponovimo eksperimente iz prejšnjega razdelka z uporabo faktorjev pozabljanja, pri čemer postavimo $\alpha := 0.995$. Dobimo enake rezultate.

Slika 9.4 prikazuje bledeče napake in pripadajočo Q-statistiko primerjave VFDT-MAJ in CVFDT-MAJ, ki uporabljata večinski klasifikator v listih. Kot prej je značilno boljši VFDT-MAJ (velja $p < 0.0001$).

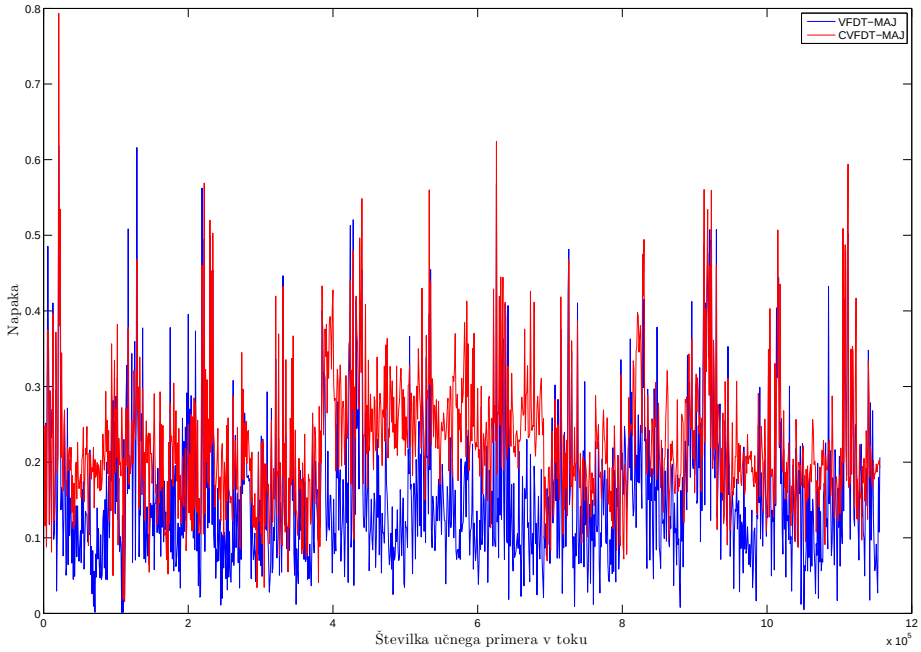
Slika 9.5 prikazuje situacijo, kjer sta algoritma VFDT-NB in CVFDT-NB skoraj enaka in ne moremo povedati, kateri je boljši, ker imamo $p = 0.1424$.

Na sliki 9.6 je primerjava algoritmov CVFDT-MAJ in CVFDT-NB, pri čemer je CVFDT-NB značilno boljši (imamo $p < 0.0001$).

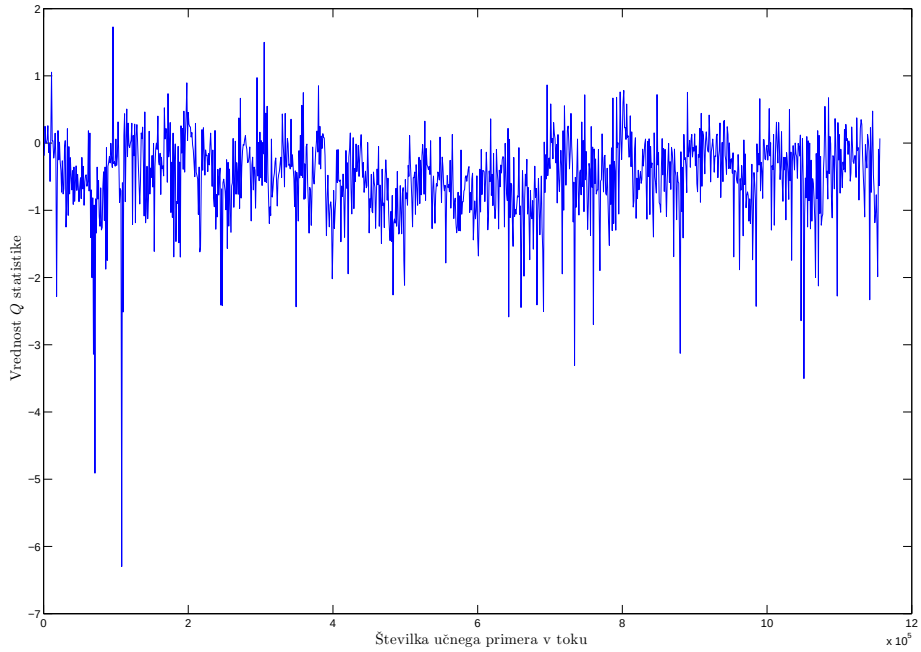
V tabeli 9.1 navajamo podrobnosti statističnih primerjav. Za vsako izmed primerjav navedemo povprečje μ, mediano $\mu_{1/2}$ in p-vrednost p.

Način	Učenec A/Učenec B	Wilcoxonov test za H_0: $Q(A, B) = 0$
Z izločanjem učnih primerov	VFDT-MAJ/CVFDT-MAJ	$\mu = -0.4954$, $\mu_{1/2} = -0.4285$, $p < 0.0001$
Z izločanjem učnih primerov	VFDT-NB/CVFDT-NB	$\mu = -0.0061$, $\mu_{1/2} = 0$, $p = 0.6538$
Z izločanjem učnih primerov	CVFDT-MAJ/CVFDT-NB	$\mu = 0.4893$, $\mu_{1/2} = 0.4410$, $p < 0.0001$
S faktorji pozabljanja	VFDT-MAJ/CVFDT-MAJ	$\mu = -0.5521$, $\mu_{1/2} = -0.377$, $p < 0.0001$
S faktorji pozabljanja	VFDT-NB/CVFDT-NB	$\mu = 0.0936$, $\mu_{1/2} = 0.0297$, $p = 0.1424$
S faktorji pozabljanja	CVFDT-MAJ/CVFDT-NB	$\mu = 0.6456$, $\mu_{1/2} = 0.3819$, $p < 0.0001$

Tabela 9.1: Rezultati Wilcoxonovega testa za testiranje hipoteze, da je mediana Q-statistik enaka nič.

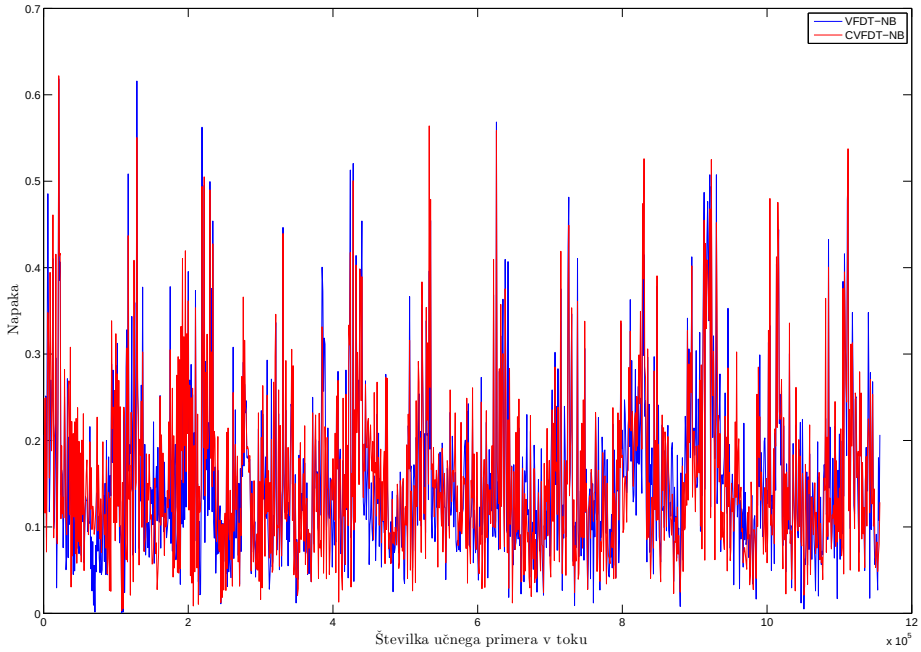


(a) Napake naših algoritmov na NYEL-NUM.

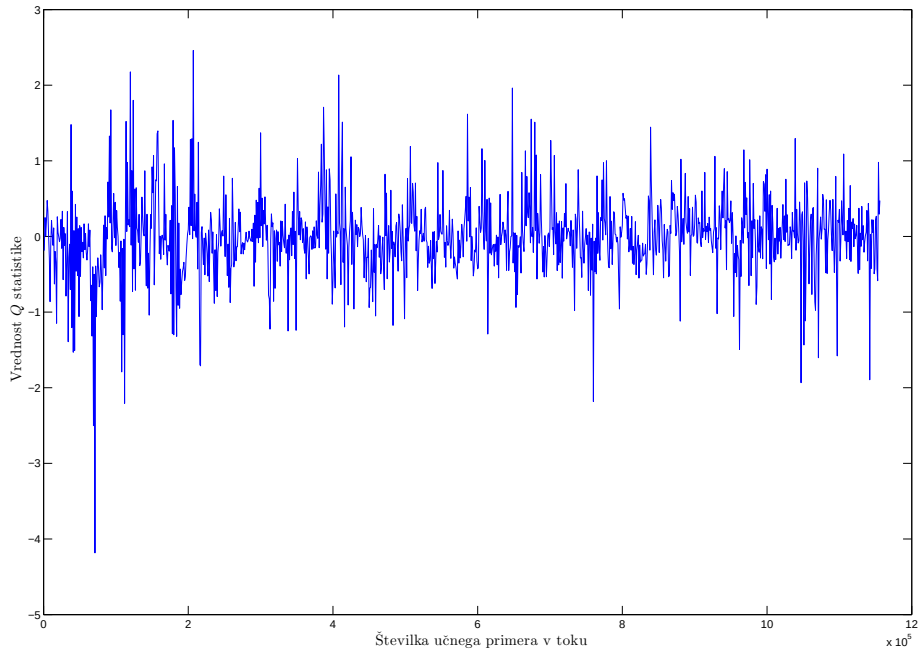


(b) Graf Q-statistike, pri čemer je A algoritem brez pozabljanja in B algoritem s pozabljanjem.

Slika 9.1: Primerjava VFDT-MAJ in CVFDT-MAJ algoritmov pri napovedovanju porabe čez 5min.

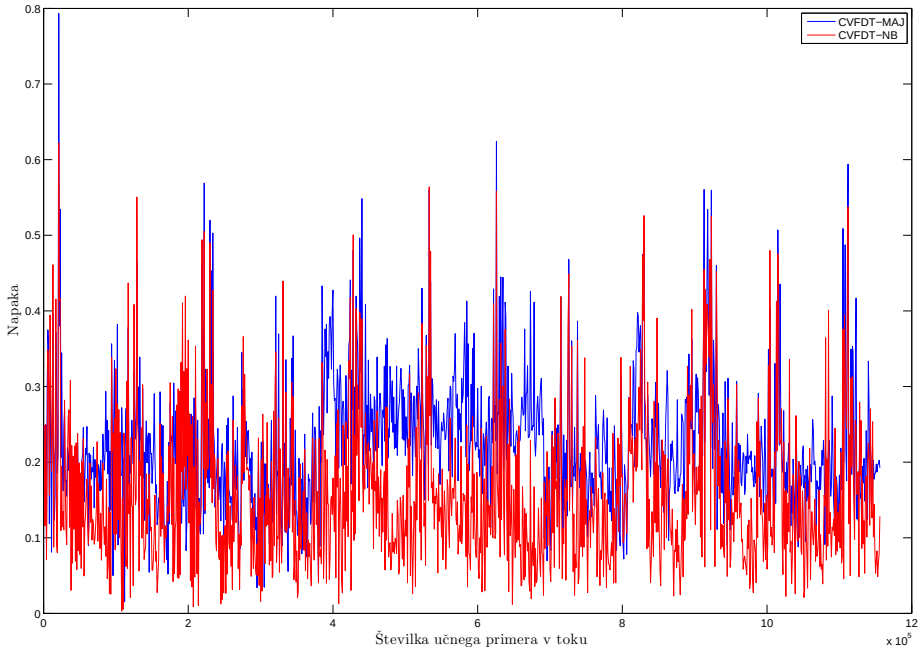


(a) Napake naših algoritmov na NYEL-NUM.

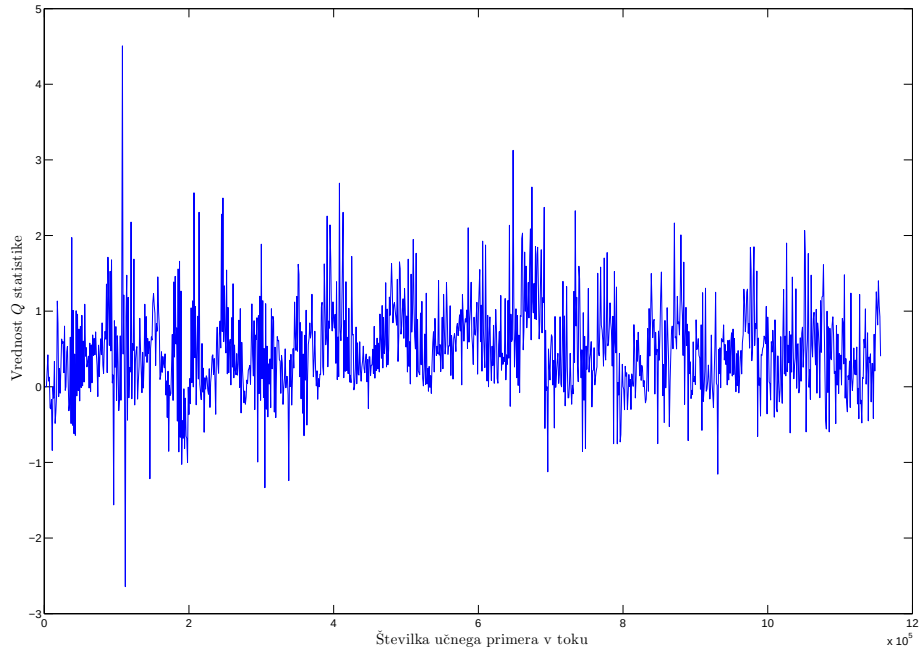


(b) Graf Q-statistike, pri čemer je A algoritem VFDT-NB in B algoritem CVFDT-NB.

Slika 9.2: Primerjava VFDT-NB in CVFDT-NB algoritmov pri napovedovanju porabe čez 5min.

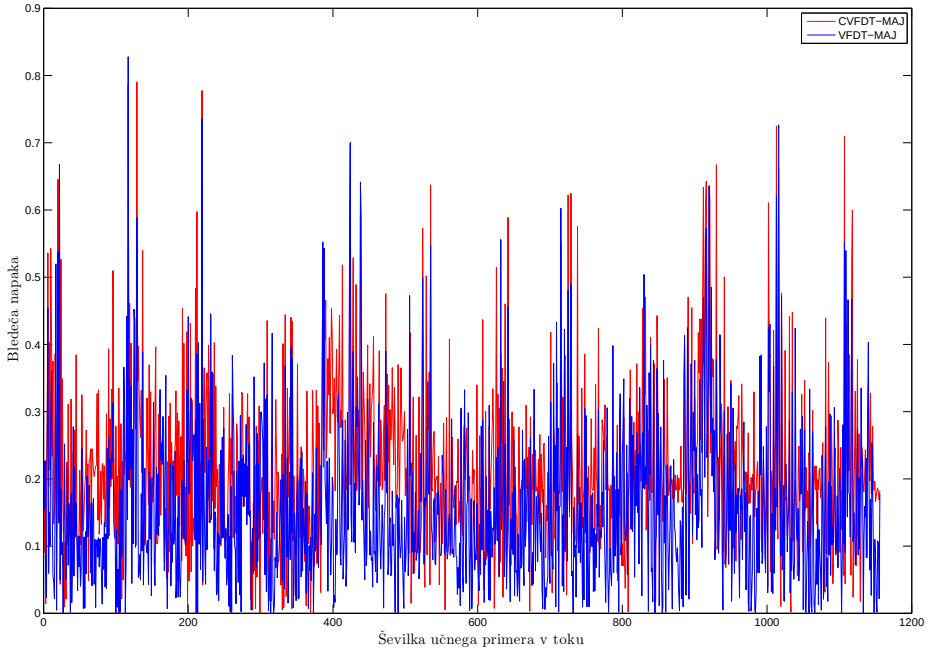


(a) Napake teh dveh algoritmov na NYEL-NUM.

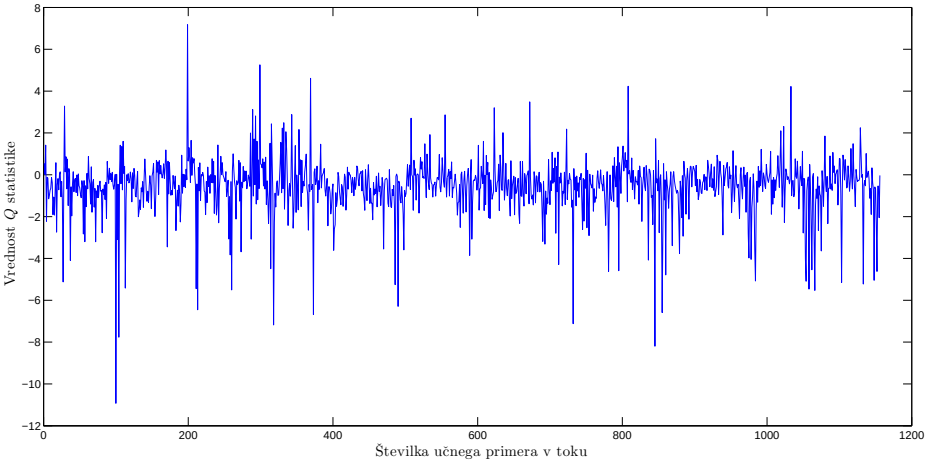


(b) Graf Q-statistike, pri čemer je A algoritem CVFDT-NB in B algoritem CVFDT-MAJ.

Slika 9.3: Primerjava CVFDT-NB in CVFDT-MAJ algoritmov pri napovedovanju porabe čez 5min.

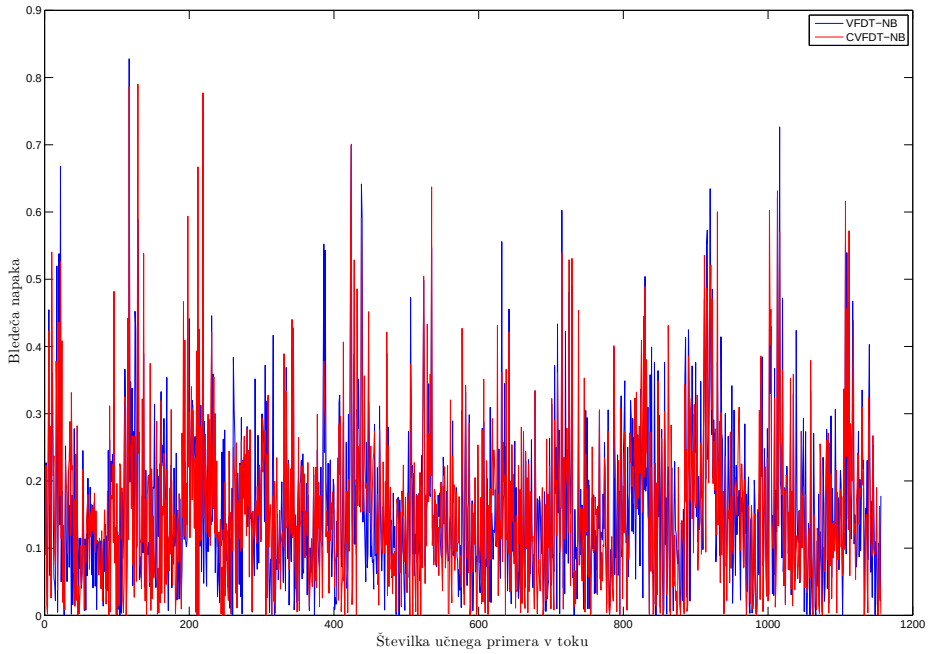


(a) Napake teh dveh algoritmov na NYEL-NUM.

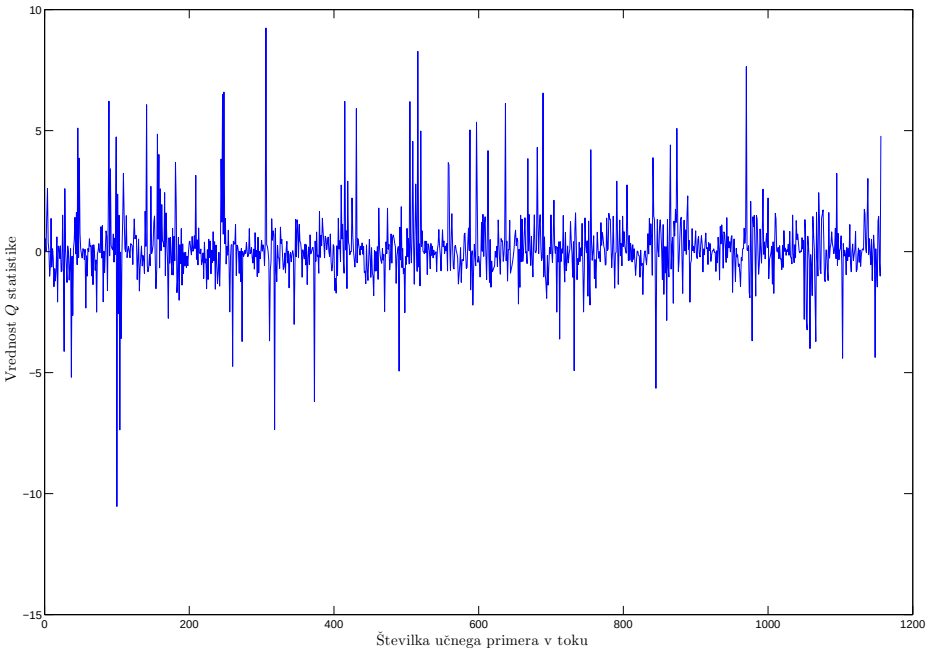


(b) Graf Q-statistike, pri čemer je A algoritem VFDT-MAJ in B algoritem CVFDT-MAJ.

Slika 9.4: Primerjava VFDT-MAJ in CVFDT-MAJ algoritmov pri napovedovanju porabe čez 5min.

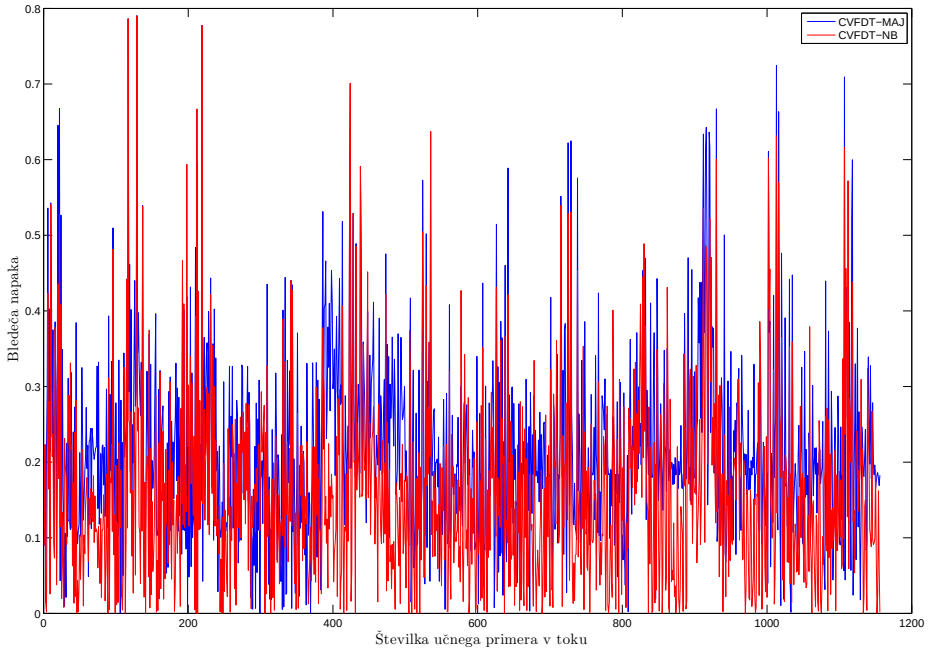


(a) Napake teh dveh algoritmov na NYEL-NUM.

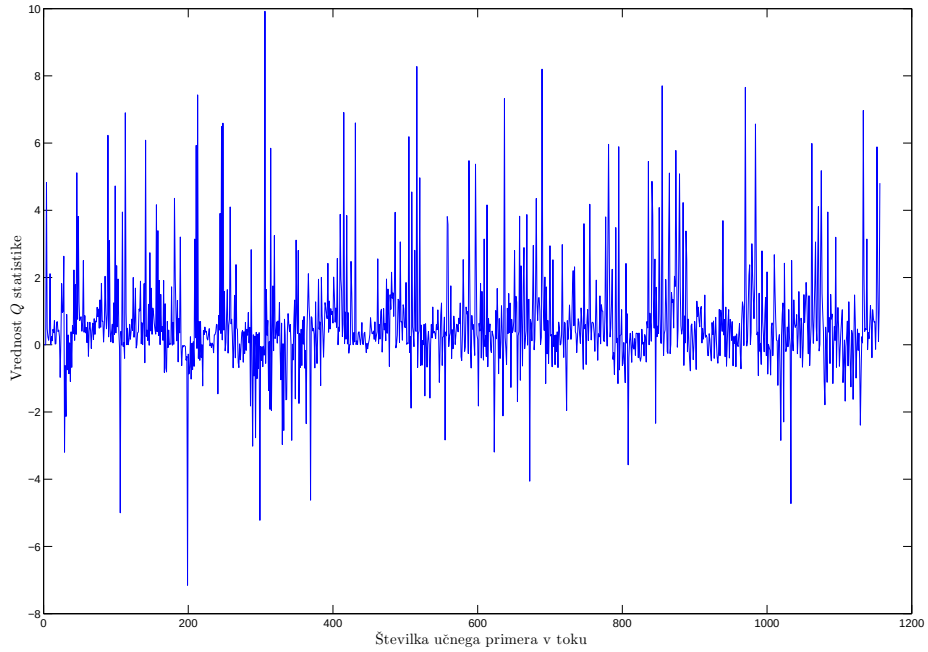


(b) Graf Q-statistike, pri čemer je A algoritem VFDT-NB in B algoritem CVFDT-NB.

Slika 9.5: Primerjava VFDT-NB in CVFDT-NB algoritmov pri napovedovanju porabe čez 5min.



(a) Napake teh dveh algoritmov na NYEL-NUM.



(b) Graf Q-statistike, pri čemer je A algoritem CVFDT-MAJ in B algoritem CVFDT-NB.

Slika 9.6: Primerjava CVFDT-MAJ in CVFDT-NB algoritmov pri napovedovanju porabe čez 5min.

Del III

Dodatki

A Implementacije

V tem poglavju navedemo pomembnejše obstoječe implementacije.

A.1 VFML

Domingos in Hulten [Hulten and Domingos, 2003] sta izdala odprtokodno orodje VFML, ki med drugim vključuje implementacije skaliranih algoritmov za učenje odločitvenih dreves, razvrščanje, in nekaterih drugih modelov. Orodje je v celoti implementirano v jeziku C, prevede se vsaj na UNIX in Windows platformah[1].

Orodje je v grobem sestavljeno iz treh delov: (i) zbirke funkcij za lažji razvoj novih algoritmov, (ii) implementacij pomembnejših učnih algoritmov, in (iii) implementacij skaliranih algoritmov, med drugim VFDT, CVFDT, in VFKM, ki smo jih omenili v pregledu dela.

Elena Ikonomovska je VFML razširila za učenje regresijskih dreves; koda je v času tega pisanja dostopna na `http://kt.ijs.si/elena_ikonomovska/FIMT-DD.zip`.

(Izpostavljamo, da za VFML obstaja spodobna dokumentacija s preprostimi primeri uporabe, kar je za odprokodno skupnost skoraj redkost.)

A.2 MOA

MOA [Bifet et al., 2010b] je — vsaj glede na število navedb v čankih — trenutno daleč najbolj popularna implementacija. Gre za naslednjika dobro poznane WEKA. Knjižnica je implementirana v Javi[2] in (verjetno edina) podpira veliko večino evalvacijskih metod iz [Gama et al., 2009, 2013] ter klasifikacijske, regresijske, in razvrščevalne metode. Je dobro dokumentirana[3] in ima aktivno uporabniško Google Groups skupino[4].

Eden od avtorjev (AB) je objavil priročnik rudarjenja podatkovnih tokov s temeljitim pregledom literature [Bifet et al., 2011].

V preprostih eksperimentih, ki jih ne vključimo, je MOA v primerjavi z našo implementacijo izredno počasna. Glavni razlog — seveda poleg Jave — je naivna implementacija nekaterih kritičnih odsekov kode v učencih.

A.3 Ostalo

Obstajajo tudi druga orodja z vtičniki (angl. *plugins*) za rudarjenje podatkovnih tokov. Omejimo se na omembo naslednjih treh:

- VEDAS so [Kargupta et al., 2004] predstavili kot mobilen in distribuiran sistem za rudarjenje podatkovnih tokov za realnočasen pregled nad vozili[5]. V času tega pisanja se zdi, da se je

[1] Za več glej `http://www.cs.washington.edu/dm/vfml/`.
[2] Za več glej `http://moa.cms.waikato.ac.nz/`.
[3] Glej `http://moa.cms.waikato.ac.nz/downloads/`.
[4] Glej `https://groups.google.com/forum/#!forum/moa-users`.
[5] Glej `http://www.csee.umbc.edu/~hillol/vedas.html` za več.

projekt znašel v sivih časih.

- SAMOA [Morales, 2013] v času tega pisanja še ni dokončno implementirana, šlo pa naj bi za večjo razširitev orodja MOA iz prejšnjega razdelka.
- RapidMiner je velik odprtokoden sistem za podatkovno rudarjenje. Rudarjenje podatkovnih tokov ponuja kot vtičnik.

B Notacija in opis podatkov

V tem poglavju navedemo tabele uporabljenih algoritmov (tabela B.5), simbolov, in podatkov ter komentiramo notacijo (razdelek B.1) in na kratko opišemo podatke (razdelek B.2), ki smo jih uporabljali za eksperimente.

B.1 Notacija

V tem razdelku fiksiramo notacijo, ki se je držimo skozi nalogo.

B.1.1 Multimnožice

Omenimo, da na množice primerov gledamo kot na multimnožice: v toku se lahko ob različnih časih pojavita enaka primera, česar s klasično množico ne moremo opisati[1]. Skozi nalogo za voljo preglednosti govorimo o množicah in uporabljamo standardno notacijo. Vseeno definirajmo pojem multimnožice in analog unije.

Definicija 4. *Multimnožica je par (A, m), pri čemer je A množica in $m_A : A \to \mathbb{N}$ funkcija iz A v množico naravnih števil $\mathbb{N} := \{1, 2, 3, \ldots\}$. Za vsak element $a \in A$ je $m(a)$ kratnost — število pojavitev — tega elementa v multimnožici.*

Unijo multimnožic S in T označimo $U := S \uplus T$ in definiramo na naraven način: unija je par (m_U, U) za $U = S \cup T$, pri čemer je preslikava $m_U : U \to \mathbb{N}$ za $u \in U$ dana z naslednjim predpisom:

$$m_U(x) := \begin{cases} m_S(u), & u \in S \text{ in } u \notin T, \\ m_T(u), & u \notin S \text{ in } u \in T, \\ m_S(x) + m_T(x), & u \in T \text{ in } u \in S. \end{cases}$$

B.1.2 Asimptotična notacija

Naj bo $f : \mathbb{R} \to \mathbb{R}^+$. Potem pišemo $f(x) = O(g(x))$ za $x \to a$, če in samo če velja

$$\limsup_{x \to a} \left| \frac{f(x)}{g(x)} \right| < \infty.$$

Podobno pišemo $f(x) = o(g(x))$ če in samo če velja

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0.$$

[1] Npr. velja $\{(1, 2, 3, Y), (1, 2, 3, Y)\} = \{(1, 2, 3, Y)\}$.

B.1.3 Ostalo

Preslikava $\operatorname{sgn} : \mathbb{R} \to \{-1, 0, 1\}$ je za $x \in \mathbb{R}$ dana s predpisom

$$\operatorname{sgn}(x) := \begin{cases} -1, & x < 0, \\ 0, & x = 0, \\ 1, & x > 0, \end{cases}$$

in nam da predznak realnega števila. Absolutna vrednost je preslikava $|\cdot| : \mathbb{R} \to \mathbb{R}$, za $x \in \mathbb{R}$ dana s predpisom

$$|x| := \begin{cases} -x, & x < 0, \\ x, & \text{sicer.} \end{cases}$$

Eksponentna funkcija je preslikava $\exp : \mathbb{R} \to \mathbb{R}^+$, za $x \in \mathbb{R}$ dana s predpisom

$$\exp(x) := \sum_{n \geq 0} \frac{x^n}{n!}.$$

Ne nekaj mestih za $x, y \in \mathbb{R}^+$ pišemo $x \approx y$, kar iz praktičnih razlogov definiramo kot

$$x \approx y \iff x \in \left[y^{\lfloor \log_{10} y \rfloor}, y^{1+\lfloor \log_{10} y \rfloor} \right],$$

kar pomeni, da je y kvečjemu za faktor 10 večji od x. Tako lahko rečemo, da je $19.34 \approx 10$, ampak $19.34 \not\approx 100$ in $19.34 \not\approx 1$. Seveda je $x \not\approx y$ natanko tedaj, ko ne velja $x \approx y$.

Simbol	Opis simbola
$\mathbb{P}$	Gostota verjetnosti
$\mathbb{E}$	Pričakovanje
$\mathbb{I}$	Indikatorska funkcija
$\mathbb{V}$	Varianca
$X \rightsquigarrow \mathcal{D}$	Konvergenca v porazdelitvi [Casella and Berger, 2002]
$[a, b] := \{x \in \mathbb{R} : a \leq x \leq b\}$	Zaprt interval
$(a, b] := \{x \in \mathbb{R} : a < x \leq b\}$	Levo-odprt interval
$[a, b) := \{x \in \mathbb{R} : a \leq x < b\}$	Desno-odprt interval
$(a, b) := \{x \in \mathbb{R} : a < x < b\}$	Odprt interval
$[n..m] := \{n, n+1, \ldots, m\}$	Množica naravnih števil od n do m
$[n] := \{1, 2, \ldots, n\}$	Množica prvih n naravnih števil
$\mathbb{R}_0^+ := \{x \in \mathbb{R} : x \geq 0\}$	Množica nenegativnih realnih števil
$\mathbb{R}^+ := \{x \in \mathbb{R} : x > 0\}$	Množica pozitivnih realnih števil
$\mathbb{N} := \{1, 2, \ldots\}$	Množica naravnih števil
$\mathbb{N}_0 := \{0, 1, 2, \ldots\}$	Množica naravnih števil z ničlo
$\left\{ {n \atop k} \right\}$	Stirlingova števila druge vrste [Graham et al., 1994]
$\binom{n}{k}$	Binomski koeficient [Graham et al., 1994]
B_n	Bellova števila [Graham et al., 1994]
$f = O(g(n))$	Veliki O notacija
$f = \Omega(g(n))$	Veliki Ω notacija
$f = \Theta(g(n))$	Veliki Θ notacija
$f = o(g(n))$	Če $f(x)/g(x) \to 0$ za $x \to \infty$
$f \sim g$	Če $f(x)/g(x) \to 1$ za $x \to \infty$
∇f	Gradient odvedljive funkcije $f : \mathbb{R}^k \to \mathbb{R}^k$

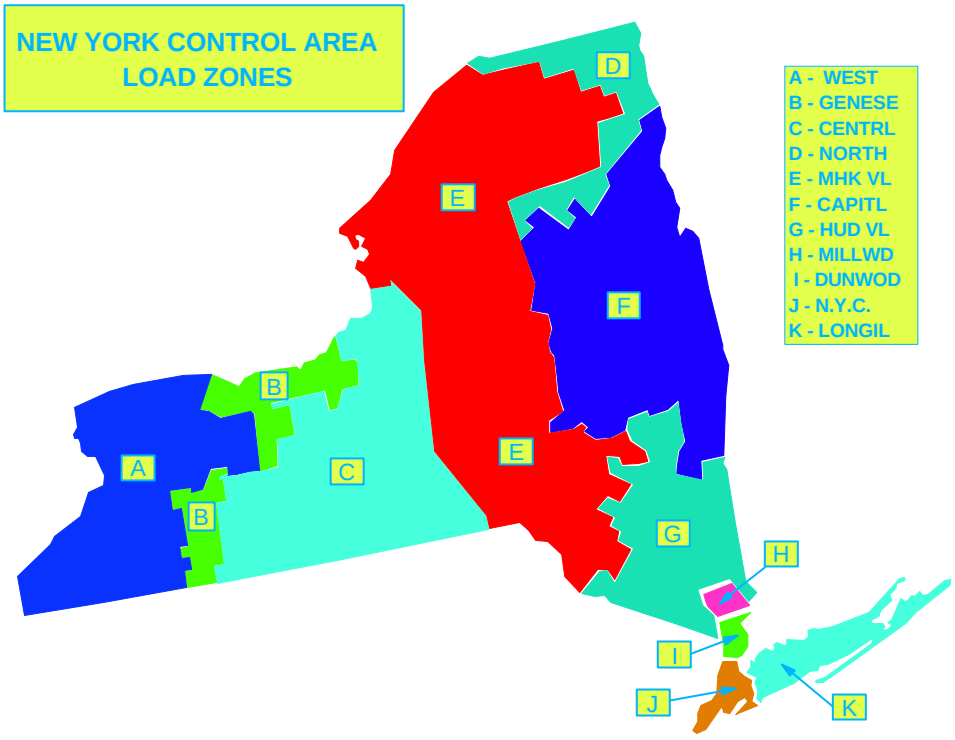
Tabela B.1: Opis uporabljenih simbolov.

B.2 Opis podatkov

V tem razdelku navedemo tabelo uporabljenih podatkov s kratkimi opisi (tabeli B.3 in B.4) in podrobneje opišemo podatke o porabi električne energije za zvezno državo New York (podrazdelek B.2.1).

B.2.1 Elektro podatki NY-EL

Realen scenarij smo pripravili na podlagi javno dostopnih[2] podatkov o porabi in ceni električne energije za zvezno državo New York, ki jih sproti objavlja neodvisni sistemski operater električnega omrežja (NYISO). Senzorji na 5 minut zajemajo vrednosti, ki so nato agregirane po območjih (glej sliko B.1)[3].



Slika B.1: Distribucijsko omrežje zvezne države New York.

V tem razdelku podrobneje opišemo, kako je bil tok NY-EL generiran na podlagi omenjenih podatkov.

Za porabo smo zajeli podatke od 26. maja 2001 do 31. decembra 2012. Vse skupaj nanese 14 418 748 (okoli 14.5M) zapisov oz. 697 500 KB (okoli 680MB) nekompresiranih podatkov. Prvih 10 zapisov:

[2] Glej `http://www.nyiso.com/public/markets_operations/market_data/pricing_data/index.jsp`.

[3] Vir: `http://www.nyiso.com/public/webdocs/markets_operations/market_data/zone_maps_graphs/nyca_zonemaps.pdf`.

Very low	Low	Normal	High (1355)	Very high (2238)
$[0, 474)$	$[474, 914.5)$	$[914.5, 1796.5)$	$[1796.5, 2238)$	$[2238, \infty)$

Tabela B.2: Diskretizacija ciljne spremenljivke.

Koda B.1: Primer porabe.

```
"Time Stamp","Time Zone","Name","PTID","Load"
"05/26/2001 00:00:00","EDT","CAPITL",61757,985
"05/26/2001 00:00:00","EDT","CENTRL",61754,1461
...
```

Za cene smo zajeli podatke od 1. januarja 2001 do 31. decembra 2012. Vse skupaj nanese 21 196 155 (okoli 21.2M) zapisov oz. 1 111 649 KB (okoli 1.1GB) nekompresiranih podatkov. Prvih 10 zapisov:

Koda B.2: Primer cen.

```
"Time Stamp","Name","PTID","LBMP ($/MWHr)","Marginal Cost Losses ($/MWHr)","
    Marginal Cost Congestion ($/MWHr)"
"01/01/2001 00:00:00","CAPITL",61757,19.51,1.38,0.00
"01/01/2001 00:00:00","CENTRL",61754,17.57,-0.56,0.00
...
```

Združili smo vrstice obeh virov podatkov, ki so se ujemale po času in po imenih. Tako dobimo 13 878 974 (okoli 13.8M) zapisov oz. 1.3GB nekompresiranih podatkov. Prvih 10 zapisov:

Koda B.3: Primer zdruzenih.

```
'"Time Stamp"', '"Time Zone"', '"Name"', '"PTID"', '"Load"', '"PTID"', '"LBMP ($/
    MWHr)"', '"Marginal Cost Losses ($/MWHr)"', '"Marginal Cost Congestion ($/MWHr)
    "'
'"05/26/2001 00:00:00"', '"EDT"', '"CAPITL"', 61757.0, 985.0, 61757.0, 20.52,
    1.06, 0.0
'"05/26/2001 00:00:00"', '"EDT"', '"CENTRL"', 61754.0, 1461.0, 61754.0, 19.0,
    -0.46, 0.0
...
```

Na podlagi datuma generiramo tri zelo informativne atribute:

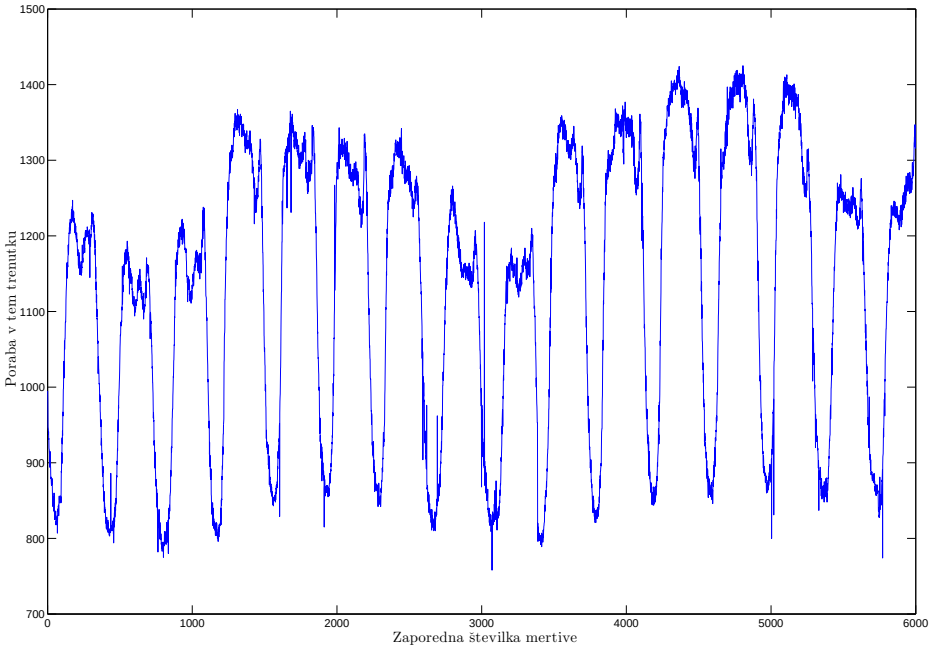
- `hourOfDay`, ki ga obravnavamo kot numeričen atribut z zalogo $[0, 24)$,
- `dayOfWeek`, ki ga obravnavamo kot numeričen atribut z zalogo $[1, 7]$,
- `month`, ki ga obravnavamo kot numeričen atribut z zalogo $[1, 12]$.

Dodamo še naslednje tri atribute:

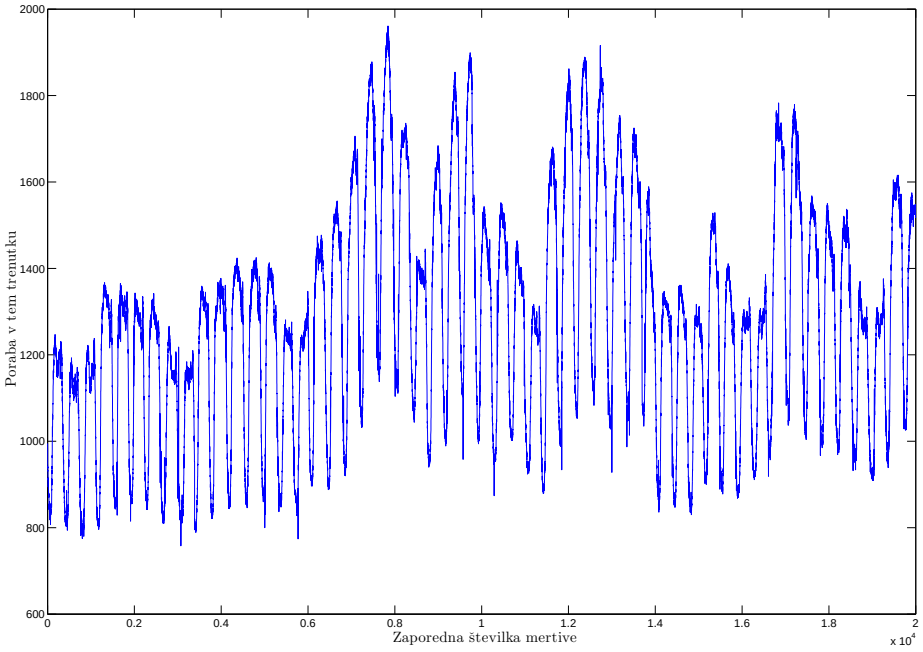
- `name`, ime države, ki ga obravnavamo kot diskreten 11-vrednosten atribut,
- `PTID`, ki ga obravnavamo kot numeričen atribut z zalogo $[6100, 6150]$,
- `load`, ki ga obravnavamo kot 5-vrednostno ciljno spremenljivko (glej tabelo B.2).

Ciljno spremenljivko smo diskretizirali na podlagi histograma porab (slika B.6). Diskretizacija je prikazana v tabeli B.2. Tem podatkom pravimo NYEL-NUM. Slednji so povzeti v tabeli B.4.

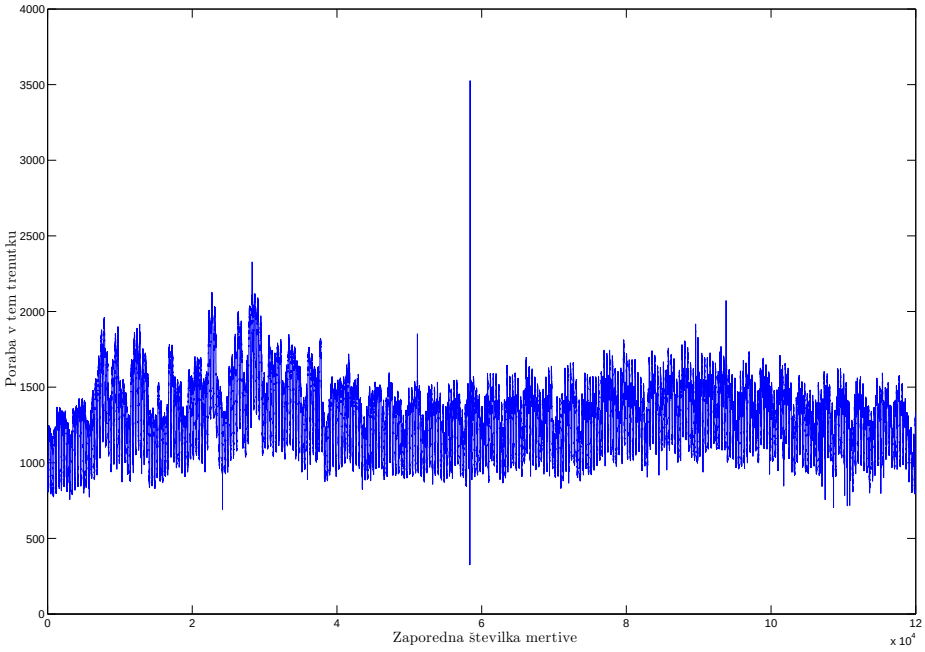
Slike B.2, B.3, B.4, in B.5 pokažejo precej regularnosti v podatkih, ki se odraža tudi v naučenih odločitvenih drevesih. Npr. ponoči in čez vikend je poraba manjša kot čez dan in čez teden.



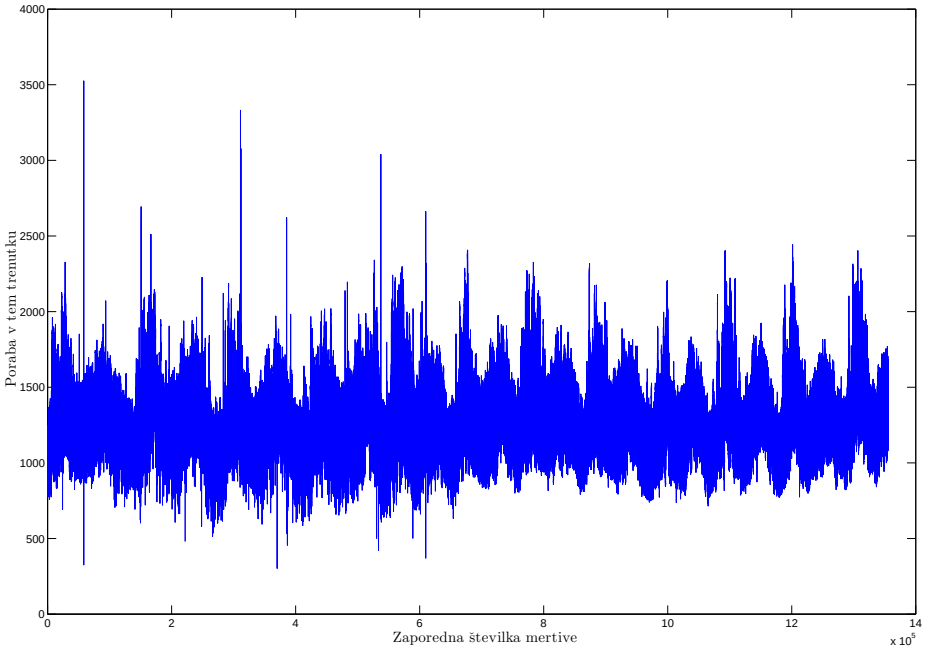
Slika B.2: Dnevni ritem porabe za 20 dni.



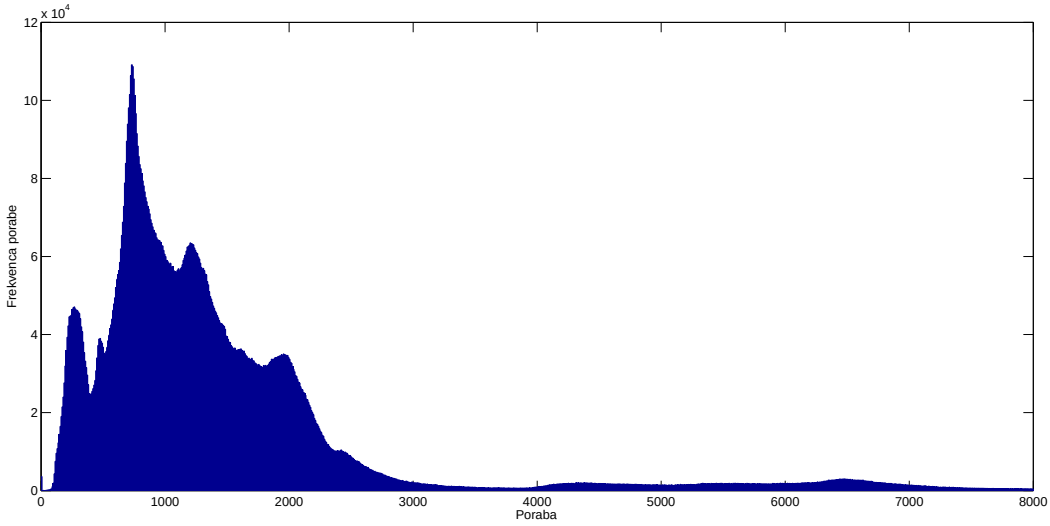
Slika B.3: Mesečni ritem porabe za pet mesecev.



Slika B.4: Letni ritem porabe.



Slika B.5: Globalni ritem porabe od januarja 2001 do decembra 2012.



Slika B.6: Histogram porab neprečiščenih podatkov.

Ime	Opis	#Primerov	#Razredov	#Nom. atr.	#Num. atr.
Titanic	Slavni Titanic podatki	2 200	2	3	0
Titanic-220K	Bootstrappan Titanic	220 000	2	3	0
Titanic-2M	Bootstrappan Titanic	2 200 000	2	3	0
Titanic-22M	Bootstrappan Titanic	22 000 000	2	3	0
Titanic-220M	Bootstrappan Titanic	220 000 000	2	3	0

Tabela B.3: Uporabljeni stacionarni podatkovni tokovi.

Ime	Opis	#Primerov	#Razredov	#Nom. atr.	#Num. atr.
Titanic-2M'	Spremenjen Titanic-2M	2 000 000	2	3	0
NYEL-NUM	Elektro podatki	13 000 000	5	2	3

Tabela B.4: Uporabljeni spremenljivi podatkovni tokovi.

Učenec	Opis učenca
VFDT-MAJ	Algoritem 1 z večinskim klasifikatorjem v listih
VFDT-NB	Algoritem 1 z Naivnim Bayesom v listih
CVFDT-MAJ	Algoritem 2 z večinskim klasifikatorjem v listih
CVFDT-NB	Algoritem 2 z Naivnim Bayesom v listih

Tabela B.5: Seznam uporabljenih učencev.

Angleški izvirnik	Prevod
intensional disagreement	neujemanje v klasifikacijskih poteh znotraj dreves
extensional disagreement	neujemanje v klasifikacijskih razredih za dani primer
multivariate delta method	multivariatna metoda delta
scaling up machine learning algorithms	vertikalno skaliranje algoritmov strojnega učenja
fading factor	bledeči faktor
fading sum	bledeča vsota
fading increment	bledeči inkrement
fading average	bledeče povprečje
prequential error	ocenjena pričakovana napaka
fading error estimator	cenilka uteženega faktorja s pozabljanjem
prequential accumulated loss	ocenjena pričakovana akumulirana cena
stream learning algorithm	algoritem za učenje iz podatkovnega toka
i.i.d.	neodvisno in enakomerno porazdeljen
holdout error	ocenjevanje z izločanjem učnih primerov

Tabela B.6: Slovar nekaterih tujk.

Literatura

Micah Adler and Brent Heeringa. Approximating optimal binary decision trees. In *Proceedings of the 11th international workshop, APPROX 2008, and 12th international workshop, RANDOM 2008 on Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques*, APPROX '08 / RANDOM '08, pages 1–9, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85362-6. doi: 10.1007/978-3-540-85363-3_1. URL http://dx.doi.org/10.1007/978-3-540-85363-3_1.

Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 81–92. VLDB Endowment, 2003.

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Boston, MA, USA, second edition, 2006. ISBN 0321486811.

Ezilda Almeida, Petr Kosina, and João Gama. Random rules from data streams. In *SAC*, pages 813–814, 2013.

Dana Angluin and Leslie G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, STOC '77, pages 30–41, New York, NY, USA, 1977. ACM. doi: 10.1145/800105.803393. URL http://doi.acm.org/10.1145/800105.803393.

Yael Ben-Haim and Elad Tom-Tov. A streaming parallel decision tree algorithm. *J. Mach. Learn. Res.*, 11:849–872, March 2010. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1756006.1756034.

Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *In SIAM International Conference on Data Mining*, 2007.

Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 139–148, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9. doi: 10.1145/1557019.1557041. URL http://doi.acm.org/10.1145/1557019.1557041.

Albert Bifet, Eibe Frank, Geoffrey Holmes, and Bernhard Pfahringer. Accurate ensembles for data streams: Combining restricted hoeffding trees using stacking. *Journal of Machine Learning Research - Proceedings Track*, 13:225–240, 2010a. URL http://dblp.uni-trier.de/db/journals/jmlr/jmlrp13.html#BifetFHP10.

Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: Massive Online Analysis. *J. Mach. Learn. Res.*, 99:1601–1604, August 2010b. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1859890.1859903.

Albert Bifet, Richard Kirkby, Geoff Holmes, and Bernhard Pfahringer. Data Stream Mining: A Practical Approach. Technical report, The University of Waikato, 2011.

Albert Bifet, Eibe Frank, Geoff Holmes, and Bernhard Pfahringer. Ensembles of restricted hoeffding trees. *ACM Trans. Intell. Syst. Technol.*, 3(2):30:1–30:20, February 2012. ISSN 2157-6904. doi: 10.1145/2089094.2089106. URL http://doi.acm.org/10.1145/2089094.2089106.

Alfred Bifet. Mining Big Data in Real Time. *Informatica*, (37):15–20, 2013. URL http://www.informatica.si/PDF/37-1/03_Bifet%20-%20Mining%20Big%20Data%20in%20Real%20Time.pdf.

Zoran Bosnić, Pedro Pereira Rodrigues, Igor Kononenko, and João Gama. Correcting streaming predictions of an electricity load forecast system using a prediction reliability estimate. In Tadeusz Czachórski, Stanisław Kozielski, and Urszula Stańczyk, editors, *Man-Machine Interactions 2*, volume 103 of *Advances in Intelligent and Soft Computing*, pages 343–350. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-23168-1. doi: 10.1007/978-3-642-23169-8_37. URL http://dx.doi.org/10.1007/978-3-642-23169-8_37.

Janez Brank. Newscluster web service, 2013. Osebna komunikacija.

G.C. Casella and Roger Berger. *Statistical Inference*. Duxbury Advanced Series. Duxbury Press, 2002. ISBN 9780495391876.

Bojan Cestnik. *Ocenjevanje verjetnosti v avtomatskem učenju*. PhD thesis, Univerza v Ljubljani, 1991.

Tony F. Chan, Gene H. Golub, and Randall J. LeVeque. Updating formulae and a pairwise algorithm for computing sample variances. Technical report, Stanford, CA, USA, 1979.

Don Coppersmith, Se June Hong, and Jonathan R. M. Hosking. Partitioning nominal attributes in decision trees. *Data Min. Knowl. Discov.*, 3(2):197–217, June 1999. ISSN 1384-5810. doi: 10.1023/A:1009869804967. URL http://dx.doi.org/10.1023/A:1009869804967.

Jaka Demšar. Razlaga napovednih modelov in posameznih napovedi pri inkrementalnem učenju, 2012. URL http://eprints.fri.uni-lj.si/1841/.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7: 1–30, December 2006. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1248547.1248548.

Pedro Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10): 78–87, October 2012. ISSN 0001-0782. doi: 10.1145/2347736.2347755. URL http://doi.acm.org/10.1145/2347736.2347755.

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM. ISBN 1-58113-233-6. doi: 10.1145/347090.347107. URL http://doi.acm.org/10.1145/347090.347107.

Pedro Domingos and Geoff Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 106–113, San Francisco, CA, USA, 2001a. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL http://dl.acm.org/citation.cfm?id=645530.658293.

Pedro Domingos and Geoff Hulten. Learning from infinite data in finite time. In *In Advances in Neural Information Processing Systems 14*, pages 673–680. MIT Press, 2001b.

Pedro Domingos and Geoff Hulten. A general framework for mining massive data stream. *Journal of Computational and Graphical Statistics*, 12:2003, 2003.

Nicholas I. Fisher and Willem R. Van Zwet. Remembering Wassily Hoeffding. *Statistical Science*, pages 536–547, 2008. URL http://arxiv.org/abs/0906.4013.

Blaž Fortuna and Jan Rupnik. QMiner. 2014. URL http://qminer.ijs.si/.

João Gama, Ricardo Rocha, and Pedro Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 523–528, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. doi: 10.1145/956750.956813. URL http://doi.acm.org/10.1145/956750.956813.

João Gama and Petr Kosina. Learning decision rules from data streams. In *IJCAI*, pages 1255–1260, 2011.

João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 329–338. ACM, 2009.

João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013.

João Gama. A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1(1):45–55, 2012. ISSN 2192-6352. doi: 10.1007/s13748-011-0002-6. URL http://dx.doi.org/10.1007/s13748-011-0002-6.

SaraA. Geer. On hoeffding's inequality for dependent random variables. In Herold Dehling, Thomas Mikosch, and Michael Sørensen, editors, *Empirical Process Techniques for Dependent Data*, pages 161–169. Birkhäuser Boston, 2002. ISBN 978-1-4612-6611-2. doi: 10.1007/978-1-4612-0099-4_4. URL http://dx.doi.org/10.1007/978-1-4612-0099-4_4.

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, Boston, MA, USA, second edition, 1994. ISBN 0201558025.

Ben J. Green. Large deviation results for combinatorics and number theory. 2013. URL https://www.dpmms.cam.ac.uk/~bjg23/papers/deviate.pdf.

Marko Grobelnik. Big Data Tutorial, 2013. URL www.slideshare.net/markogrobelnik/big-data-tutorial-marko-grobelnik-25-may-2012.

Thomas Hancock, Tao Jiang, Ming Li, and John Tromp. Lower bounds on learning decision lists and trees. *Inf. Comput.*, 126(2):114–122, May 1996. ISSN 0890-5401. doi: 10.1006/inco.1996.0040. URL http://dx.doi.org/10.1006/inco.1996.0040.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Geoff Hulten and Pedro Domingos. VFML – a toolkit for mining high-speed time-changing data streams. 2003. URL http://www.cs.washington.edu/dm/vfml/.

Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM. ISBN 1-58113-391-X. doi: 10.1145/502512.502529. URL http://doi.acm.org/10.1145/502512.502529.

Geoff Hulten, Pedro Domingos, and Laurie Spencer. Mining Massive Data Streams. Technical report, 2005. URL http://www.cs.washington.edu/dm/vfml/papers/vfdt-journal.pdf. Neobjavljeno.

Laurent Hyafil and Ronald L. Rivest. Constructing Optimal Binary Decision Trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

Elena Ikonomovska. *Algoritmi za učenje regresijskih dreves in ansamblov iz spremenljivih podatkovnih tokov*. PhD thesis, Mednarodna podiplomska sola Jožefa Stefana, 2012.

Elena Ikonomovska and Joao Gama. Learning model trees from data streams. In Jean-François Jean-Fran, MichaelR. Berthold, and Tamás Horváth, editors, *Discovery Science*, volume 5255 of *Lecture Notes in Computer Science*, pages 52–63. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-88410-1. doi: 10.1007/978-3-540-88411-8_8. URL http://dx.doi.org/10.1007/978-3-540-88411-8_8.

Elena Ikonomovska, João Gama, Raquel Sebastião, and Dejan Gjorgjevik. Regression trees from data streams with drift detection. In João Gama, VítorSantos Costa, AlípioMário Jorge, and PavelB. Brazdil, editors, *Discovery Science*, volume 5808 of *Lecture Notes in Computer Science*, pages 121–135. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04746-6. doi: 10.1007/978-3-642-04747-3_12. URL http://dx.doi.org/10.1007/978-3-642-04747-3_12.

Elena Ikonomovska, João Gama, and Sašo Džeroski. Learning model trees from evolving data streams. *Data Min. Knowl. Discov.*, 23(1):128–168, July 2011. ISSN 1384-5810. doi: 10.1007/s10618-010-0201-y. URL http://dx.doi.org/10.1007/s10618-010-0201-y.

Ruoming Jin and Gagan Agrawal. Efficient decision tree construction on streaming data. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 571–576, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. doi: 10.1145/956750.956821. URL http://doi.acm.org/10.1145/956750.956821.

M.J. Jones and P. Viola. Robust real-time object detection. In *Workshop on Statistical and Computational Theories of Vision*, 2001.

Hillol Kargupta, Ruchita Bhargava, Kun Liu, Michael Powers, Patrick Blair, Samuel Bushra, James Dull, Kakali Sarkar, Martin Klein, Mitesh Vasa, and David Handy. VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring. SIAM, 2004. URL http://www.siam.org/meetings/sdm04/proceedings/sdm04_028.pdf.

Grega Kešpret. Ocenjevanje zanesljivosti napovedi pri regresijskem napovedovanju iz podatkovnih tokov, 2012. URL http://eprints.fri.uni-lj.si/1759/.

Donald E. Knuth. *Combinatorial Algorithms*, volume 4A of *The Art of Computer Programming*. Addison-Wesley Professional, first edition, January 2011.

Igor Kononenko and Marko Robnik-Šikonja. *Inteligentni Sistemi*. Založba FE in FRI, 2010.

Petr Kosina and João Gama. Handling time changing data with adaptive very fast decision rules. In *ECML/PKDD (1)*, pages 827–842, 2012a.

Petr Kosina and João Gama. Very fast decision rules for multi-class problems. In *SAC*, pages 795–800, 2012b.

Eduardo S. Laber and Loana Tito Nogueira. On the hardness of the minimum height decision tree problem. *Discrete Appl. Math.*, 144(1-2):209–212, November 2004. ISSN 0166-218X. doi: 10.1016/j.dam.2004.06.002. URL http://dx.doi.org/10.1016/j.dam.2004.06.002.

Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

Gianmarco De Francisci Morales. SAMOA: A platform for mining big data streams, 2013.

Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995. ISBN 0-521-47465-5, 9780521474658.

Blaž Novak. Odkrivanje tematik v zaporedju besedil in sledenje njihovim spremembam, 2008. URL http://agava.ijs.si/~blazn/diploma/diploma.pdf.

Blaž Novak. Efficient Clustering of Document Streams, 2009. Neobjavljeno.

Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. Handling Numeric Attributes in Hoeffding Trees. In Takashi Washio, Einoshin Suzuki, KaiMing Ting, and Akihiro Inokuchi, editors, *Advances in Knowledge Discovery and Data Mining*, volume 5012 of *Lecture Notes in Computer Science*, pages 296–307. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-68124-3. doi: 10.1007/978-3-540-68125-0_27. URL http://dx.doi.org/10.1007/978-3-540-68125-0_27.

J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

Jesse Read, Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Streaming multi-label classification. *Journal of Machine Learning Research - Proceedings Track*, 17:19–25, 2011. URL http://dblp.uni-trier.de/db/journals/jmlr/jmlrp17.html#ReadBHP11.

Jesse Read, Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Scalable and efficient multi-label classification for evolving data streams. *Mach. Learn.*, 88(1-2):243–272, July 2012. ISSN 0885-6125. doi: 10.1007/s10994-012-5279-6. URL http://dx.doi.org/10.1007/s10994-012-5279-6.

Pedro Pereira Rodrigues, Zoran Bosnić, João Gama, and Igor Kononenko. Estimating Reliability for Assessing and Correcting Individual Streaming Predictions. In *Reliable Knowledge Discovery*, pages 29–49. Springer US, 2012.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, November 1958.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, third edition, 2012. ISBN 0137903952. URL http://portal.acm.org/citation.cfm?id=773294.

Steven L. Salzberg. On comparing classifiers: Pitfalls toavoid and a recommended approach. *Data Min. Knowl. Discov.*, 1(3):317–328, January 1997. ISSN 1384-5810. doi: 10.1023/A:1009752403260. URL http://dx.doi.org/10.1023/A:1009752403260.

André Schlichting and Blaž Sovdat. Incremental entropy computation. MathOverflow, 2013. URL http://mathoverflow.net/questions/133986. [Online; accessed 2013-06-17].

JeffreyC. Schlimmer and Jr. Granger, RichardH. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986. ISSN 0885-6125. doi: 10.1007/BF00116895. URL http://dx.doi.org/10.1007/BF00116895.

Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948. URL http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf.

Detlef Sieling. Minimization of decision trees is hard to approximate. *Journal of Computer and System Sciences*, 74(3):394–403, 2008.

Blaž Sovdat. Updating Formulas and Algorithms for Computing Entropy and Gini Index on Time-Changing Data Streams. 2013. URL http://arxiv.org/abs/1403.6348.

Terence Tao. *Topics in random matrix theory*. 2011. URL http://terrytao.files.wordpress.com/2011/02/matrix-book.pdf.

P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511 – I–518 vol.1, 2001. doi: 10.1109/CVPR.2001.990517.

Larry Wasserman. *All of Nonparametric Statistics (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387251456.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *ACM SIGMOD Record*, volume 25, pages 103–114. ACM, 1996.

Slike

Tabele

Algoritmi

Kodni izseki